A Survey of Hierarchical Energy Optimization for Mobile Edge Computing: A Perspective from End Devices to the Cloud

PEIJIN CONG, East China Normal University, China
JUNLONG ZHOU, Nanjing University of Science and Technology, China
LIYING LI, KUN CAO, and TONGQUAN WEI, East China Normal University, China
KEQIN LI, FELLOW, IEEE, State University of New York

With the development of wireless technology, various emerging mobile applications are attracting significant attention and drastically changing our daily lives. Applications such as augmented reality and object recognition demand stringent delay and powerful processing capability, which exerts enormous pressure on mobile devices with limited resources and energy. In this article, a survey of techniques for mobile device energy optimization is presented in a hierarchy of device design and operation, computation offloading, wireless data transmission, and cloud execution of offloaded computation. Energy management strategies for mobile devices from hardware and software aspects are first discussed, followed by energy-efficient computation offloading frameworks for mobile applications that trade application response time for device energy consumption. Then, techniques for efficient wireless data communication to reduce transmission energy are summarized. Finally, the execution mechanisms of application components or tasks in various clouds are further described to provide energy-saving opportunities for mobile devices. We classify the research works based on key characteristics of devices and applications to emphasize their similarities and differences. We hope that this survey will give insights to researchers into energy management mechanisms on mobile devices, and emphasize the crucial importance of optimizing device energy consumption for more research efforts in this area.

CCS Concepts: • **Networks** → **Network mobility**; *Cloud computing*;

Additional Key Words and Phrases: Computation offloading, energy optimization, mobile devices, mobile computing (MC), mobile cloud computing (MCC), mobile edge computing (MEC), wireless communication

This work was supported in part by National Key Research and Development Program of China under Grant 2018YFB2101300, ECNU xingfuzhihua Program, National Natural Science Foundation of China under Grants 61802185 and 61872147, Natural Science Foundation of Jiangsu Province under Grant BK20180470, and the Fundamental Research Funds for the Central Universities under Grant 30919011233.

Authors' addresses: P. Cong, L. Li, K. Cao, and T. Wei (corresponding author), East China Normal University, School of Computer Science and Technology, 3663 Zhongshan North Road, Shanghai, China; emails: 52184506011@stu.ecnu.edu.cn, 874023104@qq.com, 52174506005@stu.ecnu.edu.cn, tqwei@cs.ecnu.edu.cn; J. Zhou, Nanjing University of Science and Technology, School of Computer Science and Engineering, Xiaolingwei 200#, Nanjing, China; email: jlzhou@njust.edu.cn; K. Li, State University of New York, Department of Computer Science, 1 Hawk Drive, New Paltz, New York; email: lik@newpaltz.edu.

0360-0300/2020/04-ART38 $15.00
https://doi.org/10.1145/3378935

ACM Reference format:
Peijin Cong, Junlong Zhou, Liying Li, Kun Cao, Tongquan Wei, and Keqin Li, Fellow, IEEE. 2020. A Survey of Hierarchical Energy Optimization for Mobile Edge Computing: A Perspective from End Devices to the Cloud. *ACM Comput. Surv.* 53, 2, Article 38 (April 2020), 44 pages.
https://doi.org/10.1145/3378935

1 INTRODUCTION

Mobile devices have experienced incredible changes in the past few decades, starting from devices with single function like voice services to devices providing diverse applications and services to users. Mobile devices are computing devices made for portability, thus are small and lightweight enough to operate in the hand, bringing great convenience to people's daily lives [MD 2019]. Based on statistics from Statista, the number of worldwide mobile device users is expected to exceed five billion in 2019 [sta 2018]. However, performing computing with mobile devices brings undeniable challenges that cannot be ignored, such as memory, processing, battery lifetime, compared with their static counterparts [Namboodiri and Ghose 2012]. The discrepancy in growth rate between the batteries' volumetric/gravimetric energy density and the power requirements brings unprecedented challenges to resource-constrained mobile devices [Lin et al. 2014].

From the perspective of device hardware, take a smartphone as an example, it usually contains the following necessary components in a mobile operating system, such as processors, memory, sensors, battery, display, and wireless network interfaces. Most of these components are especially power hungry to inadvertently drain out the battery energy quickly. For example, the processors are considered to be one of the most energy-intensive components on mobile devices [Tian et al. 2018]. From the perspective of the applications running on mobile operating systems (OSs), with the ubiquitousness of mobile devices such as smartphones, tablets, and IoT devices, numerous applications have emerged and attracted great attention. These novel applications, such as mobile games, augmented reality, and object recognition, are all computational and energy-consuming. Taking into account the physical size constraint on mobile devices, these applications are always restricted by mobile devices' limited resources and battery power [Guo and Liu 2018].

Mobile cloud computing (MCC), edge computing (EC), and fog computing (FC) as new paradigms open up the possibility of energy-saving opportunities for mobile devices by offloading computation, communication, or resource-intensive application components to powerful servers for execution. Offloading is a process of data migration from mobile devices to an external computing platform, such as a cluster, grid, or cloud [CO 2019]. MCC utilizes powerful servers to replace mobile devices for processing power-consuming tasks to prolong the battery lifetime. However, cloud servers are spatially far from mobile devices; thus, offloading tasks to remote clouds may incur significant transmission energy consumption. Fortunately, mobile edge computing (MEC) and FC can efficiently solve this problem by making the abundant resources closer to mobile devices, thus decreasing the application execution latency and transmission energy consumption. However, though offloading can reduce the burden of mobile devices, it is a complicated technology when performing offloading taking into account various factors, e.g., application partition, network conditions, and cloud availability.

Several surveys on MC have been published in the recent past. However, few of them investigate advanced energy optimization techniques of mobile devices. Welch [1995] only briefly surveyed several research works that reduce energy consumption of a few hardware components (e.g., CPU and hard drives) on mobile devices. By comparison, Rodriguez et al. [2013] provided a detailed review of software optimization for energy management on mobile devices. Nevertheless, the energy optimization for hardware components and cloud-assisted offloading strategies for mobile devices are not fully explored in this work. Kumar et al. [2013] discussed various offloading strategies to

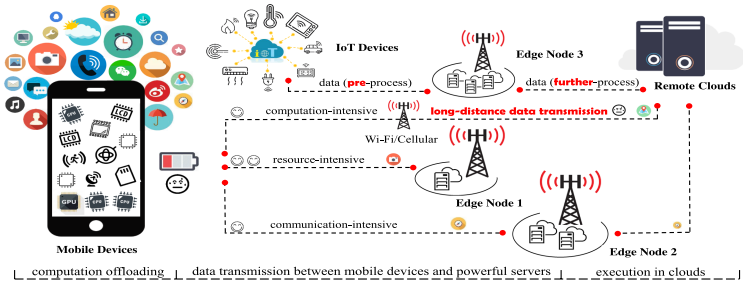


Fig. 1. A bird's-eye view of the central idea of this article.

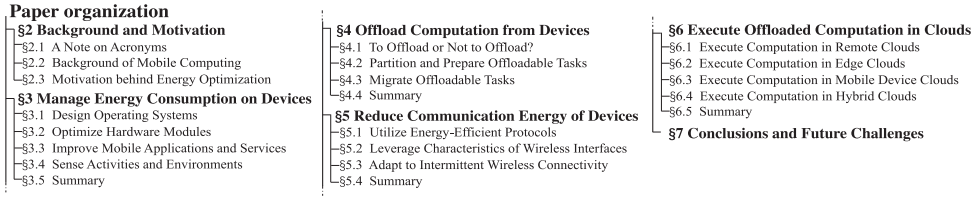


Fig. 2. Organization of the article in different sections.

optimize device energy consumption. However, the significant wireless communication energy of mobile devices incurred by offloading is not considered in their work. Cui et al. [2013] save device energy by only investigating the techniques on energy-efficient wireless transmission. In general, all aforementioned works describe the energy optimization methods for mobile devices from only one angle. These surveys do not provide a comprehensive investigation of mobile device energy optimization techniques.

Contribution and article organization. In this article, a survey of mobile device energy optimization techniques is presented. Figure 1 provides a bird's-eye view of the central idea of this article. Figure 2 gives the organization of the article in different sections.

We first discuss the background and motivations for energy optimization of mobile devices (Section 2). We then review techniques for energy management of a mobile device from perspectives of both hardware and software (Section 3). In Section 4, we compare computation offloading strategies for energy savings by migrating computation-intensive application components to powerful servers for execution. We discuss research works on how to optimize wireless communication energy incurred by computation offloading in Section 5. Further, we summarize works on application and task execution mechanisms in clouds (Section 6). Concluding remarks of this article followed by a discussion on future challenges are presented in Section 7.

Scope of the article. The scope of device energy optimization covers a broad range of techniques, including MC, MCC, and MEC. For ease of presentation, the scope of this article is limited in the following way. We concentrate on works that optimize device energy consumption. We do not include works that focus on device performance improvement or other goals. We hope this article can be a valuable resource for developers, mobile OS designers, and other researchers.

2 BACKGROUND AND MOTIVATION

2.1 A Note on Acronyms

In this section, the list of acronyms used in this article are summarized in Table 1 for a concise presentation.

Table 1. List of Acronyms

Symbol	Description	Symbol	Description
MC	Mobile Computing	CC	Cloud Computing
EC	Edge Computing	FC	Fog Computing
MCC	Mobile Cloud Computing	MEC	Mobile Edge Computing
3G/4G/5G	Third/Fourth/Fifth Generation	IoT	Internet of Things
RAN	Radio Access Network	NFC	Near-Field Communication
MDC	Mobile Device Cloud	OS	Operating System
MPSoC	Multiprocessor System-on-Chip	DVFS	Dynamic Voltage and Frequency Scaling
FPS	Frames Per Second	DAG	Directed Acyclic Graph
TIG	Task Interaction Graph	IPC	Inter-Process Communication
RPC	Remote Procedure Call	HW	Hardware
VMM	Virtual Machine Manager	AP	Access Point
DNN	Deep Neural Network	IP	Internet Protocol Address
CNN	Convolutional Neural Network	MDP	Markov Decision Process

2.2 Background of Mobile Computing

The success of the father of radio, Guglielmo Marconi [2019], in producing radio waves over long distances marked the beginning of wireless network technologies. The advances in wireless network technologies have further brought about a new computing paradigm, that is, MC, which is a computing paradigm allowing delivery of data via wireless-enabled mobile devices without needing to connect to a fixed physical link. Users carrying these devices can access various services via a shared infrastructure. MC usually refers to the cooperation between mobile communication, mobile hardware, and mobile software [mc 2019]. Mobile communication refers to the infrastructure that guarantees seamless and reliable communication between devices. Mobile hardware refers to the device components that are responsible for receiving or accessing services. Mobile software refers to the programs running on mobile hardware to process demands from applications. The three components work collaboratively to provide services to users. However, the power crisis of mobile devices also raises more concerns for them. Next, we give a brief introduction to two crucial computing paradigms promoted by virtual technologies, namely, MCC and MEC, which bring new energy-saving opportunities to mobile devices with limited resources.

MCC. MCC, also called MoClo, combines the techniques of cloud computing (CC), MC, and wireless communication to provide bountiful computing and storage resources to mobile users [Liu et al. 2013a]. CC is an information technology that delivers shared pools of system resources (e.g., processors, software, and storage) over communication networks [Cong et al. 2018; Wang et al. 2020]. By leveraging this computing paradigm, mobile applications can be developed and hosted in the context of MCC. This technique enables developers to design mobile applications for users without needing to consider the power and resource limits in mobile devices. In other words, the processes of data storage and data processing are happening in an infrastructure outside of mobile devices. Mobile users can leverage various CC platforms instead of power-limited mobile devices to execute a variety of computation-intensive applications or services.

MEC. MEC refers to the technique allowing tasks to be executed closer to mobile users. MEC is pushed forward by the rapid developments of MCC, in which mobile devices use a variety of resources in remote clouds through radio access networks, which may require significant transmission energy for data transferring. MEC frees MCC from this problem by deploying resources closer to mobile devices. In this context, the data produced by mobile devices is allowed to be

Table 2. A Summary of Crucial Computing Paradigms Based on Requirement Characteristics

Requirements	CC	MCC	EC	FC	MEC
Latency	High	High	Low	Low	Low
Location of service	Network core	Network core	Network edge	Network edge	Network edge
Location awareness	No	No	Yes	Yes	Yes
Deployment	Centralized	Centralized	Distributed	Distributed	Distributed
Mobility	No	Yes	Yes	Yes	Yes
Security	No	No	Yes	Yes	Yes

processed at edge servers, instead of sending it to remote servers for execution [Shi et al. 2016]. Moreover, mobile devices not only play the role of data consumers but also serve as data producers. In this way, mobile devices can not only use services (e.g., watching a YouTube video) in edge servers but also produce various data (e.g., taking photos). MEC provides various services including computing offloading, data storage, and processing for mobile users.

Table 2 summarizes computing paradigms' requirement characteristics. CC and MCC are deployed with centralized data centers to provide services to users. In centralized computing architecture, processing/computing is performed more on central servers. On the contrary, EC and FC are distributed computing paradigms in which computation is performed on geographically distributed device nodes, making resources and services closer to users [EC 2019]. Due to high concentration and unified management of resources in CC, data security and privacy could be guaranteed to some extent. However, edge or fog servers may not guarantee data security and privacy since these distributed server nodes are more vulnerable to threats and attacks. For latency-sensitive applications, CC and MCC paradigms with high latency and long transmission distance are not suitable choices. These two computing paradigms also lack location-awareness, thus are only suitable for indoor personal computer users. EC, FC, and MEC overcome these problems by providing distributed and localized services to users. In this way, users can enjoy high-rate local connections without the need to search over the remote clouds [Luan et al. 2015]. Moreover, considering high user mobility in the contexts of EC, FC, and MEC, it is necessary to develop mobility management schemes to guarantee service continuity when mobile users roam across different edge/fog/mobile edge base-stations.

2.3 Motivation Behind Energy Optimization

Wireless devices such as smartphones, tablets, and IoT devices enable people to access network services whenever and wherever possible. However, people are increasingly concerned about devices' energy when enjoying the great convenience brought by these devices. MCC and MEC paragrams provide new insights to augment device capabilities and prolong their battery lifespan by offloading computational parts of an application to remote servers or nearby edge/fog servers for execution. However, offloading is a complicated process especially in dynamic wireless network environment [Chun and Maniatis 2010]. Before offloading, it is necessary to judiciously determine whether, what, when, and where to offload tasks under intermittent wireless channels to achieve device energy savings. Next, we analyze the energy optimization challenges for mobile devices from perspectives of device hardware and software techniques, computation offloading strategies, wireless transmission techniques, and execution mechanisms in clouds, respectively.

Mobile hardware and software. Mobile hardware and software are integrated in devices to provide services by connecting to the Internet via Bluetooth, cellular networks, or WiFi. Mobile hardware usually consists of processors, memory, battery, display, sensors, and the like. Mobile software includes mobile OSs (e.g., Android, iOS, and Windows Phone) and applications or services

running on OSs. Powered by a lithium battery and with the support of mobile hardware, mobile devices can run mobile OSs to permit applications or services to be installed to satisfy users' requirements. However, due to limited battery lifetime, users need to recharge their devices frequently, which seriously deteriorates user experience. Managing device energy consumption is imperative but challenging considering the following factors. First, energy-efficient device resources management for applications' use is important. An energy-aware OS is necessary for mobile devices to efficiently manage and control applications' use of resources [Chu et al. 2011]. Second, some components usually consume more energy than others, such as, processors, sensors, and display. Meanwhile, the thermal coupling effects existing among processors cannot be ignored when optimizing energy. Third, considering increasingly complex mobile applications such as power-hungry deep neural networks that are hard to deploy in resource-constraint mobile devices, energy optimized methods for these specific applications are also needed. Finally, it is necessary to achieve energy-efficient long-term sensing of user behaviors since more and more applications need to monitor and recognize users' activities continuously.

Computation offloading strategies. Enabled by virtualization techniques, energy constraints on mobile devices can be alleviated by offloading computational tasks of an application to clouds for execution [Lin et al. 2013; Wu and Wolter 2015]. In particular, a mobile application can be partitioned with a different granularity level (i.e., coarse [Shiraz and Gani 2014] or fine [Ge et al. 2012] grain). The partitioned parts of the mobile application are offloaded to powerful computing resources (e.g., cloud servers and edge servers) for execution. However, Altamimi et al. [2012] and Nguyen et al. [2016] argued that before implementing offloading techniques, it is necessary to judge whether computation offloading is energy-efficient for mobile devices considering various factors such as fluctuant wireless network connections. Thus, to fulfill the prospective energy-saving benefits of computation offloading on mobile devices, the following several points need to be considered. First, consider whether computation offloading is an effective solution for prolonging the battery lifetime. Sometimes, offloading for remote execution may be meaningless compared with local execution due to incurred transmission energy costs. Second, consider which portion of an application should be sent to the clouds once the offloading decisions are made. Finally, consider how to judiciously make offloading decisions (i.e., static or dynamic) for offloading computational tasks of an application to the clouds under the intermittent wireless network environment.

Wireless communication techniques. Wireless refers to communication or transmission of data between two or more nodes that are not connected by a fixed physical link [wir 2019]. Data is sent through the air by using electromagnetic waves in a wireless network environment. Most mobile devices have built in wireless network interfaces (e.g., 3G, 4G, Bluetooth, and WiFi) that can be used to reach the Internet. The offloaded computational tasks can be sent to powerful clouds for processing through these wireless interfaces. Kalic et al. [2012] found that different communication technologies usually consume different amounts of energy. Thus, understanding the relationship between energy consumed and transferred data amount is necessary when using 3G, 4G, Bluetooth, and WiFi, respectively. Compared with good wireless connectivity, mobile devices usually consume more energy when transmitting data under bad wireless connections [Shu et al. 2013]. In this way, offloading is not a feasible scheme since the extra communication overheads exceed the saved local energy. Thus, when performing offloading, both the energy consumption of wireless interfaces and the dynamic characteristics of wireless network channels need to be considered.

Execution mechanisms in clouds. MCC provides mobile devices with powerful offloading sites (e.g., servers) or device clone (e.g., CloneCloud [Chun et al. 2011] and ThinkAir [Kosta et al. 2012]) in remote clouds for executing offloaded tasks. However, the offloading process may fail considering long distance between devices and remote clouds, and unstable wireless connections. MEC offers a new perspective to provide offloading sites more near to mobile devices, such as edge

servers (e.g. cloudlets [Jararweh et al. 2014; Satyanarayanan et al. 2009] and fog servers [Luan et al. 2015]) that co-locate with cellular base-stations. In this way, the computational tasks can be offloaded to nearby edge servers rather than remote cloud servers to save transmission energy. In addition to powerful cloud servers and edge servers, Miluzzo et al. [2012] further presented their vision in which mobile devices are playing an indispensable role in MCC. Specifically, they envision a computing platform called mobile device cloud in which task migrationhas occurred between nearby devices. In this way, the offloaded tasks can be performed among nearby devices to save transmission energy when wireless connections are intermittent and unavailable.

3 MANAGE ENERGY CONSUMPTION ON DEVICES

The capacity of mobile devices is severely restricted by limited storage and computing resources as well as the fast drained battery power [Pathak et al. 2011]. It is essential to optimize energy consumption of the energy-consuming entities and the applications or services running on OSs to prolong the battery lifetime. Mobile OSs (e.g., Android, iOS, Windows 10 Mobile, and BlackBerry 10) running on mobile devices (e.g., smartphones and tablet) are designed to provide services to users. These OSs have been fully exploited and developed to help people better understand the energy usage of their devices. However, they still have much room for improvement in the aspects of both energy usage control for mobile applications running on them and proactive resources management based on user's habits and customs (Section 3.1). Mobile hardware usually includes a wireless inbuilt modem for data connection, display, cellular, Bluetooth, WiFi, Global Positioning System (GPS), cameras, processors, memory, and near-field communication (NFC) [MOS 2019]. Thereinto, different mobile modules or components consume different amounts of energy, and contribute to battery life exhaustion in different levels. For example, processors (e.g., CPUs, GPUs, and application processors), memory, sensors (e.g., GPS, accelerometer, and touchscreen sensors), and display are considered the most power consumers among mobile hardware. Thus, exploring the characteristics and operating modes of these modules or components can better optimize their energy consumption (Section 3.2).

Mobile applications running on mobile OSs provide innovative services for users. However, while mobile hardware and OS vendors have made positive progress to make mobile platforms more energy-efficient at all levels, these improvements cannot stop poorly designed applications from unnecessarily draining the battery lifetime. Some existing techniques can help developers understand the energy usage patterns of applications. Even so, they do not provide guidance about how to save energy for applications. Thus, it is necessary to design energy-efficient mobile applications or services, and develop customized energy optimization methods for these computation/memory/communication-intensive mobile applications or services running on resource/power-limited mobile devices (Section 3.3). Aside from mobile devices and applications, mobile users and ambient environments also play an important role in optimizing device energy consumption. Mobile devices always interact with their users and surrounding environments. On one hand, users' behaviors (i.e., zooming and scrolling) demand a different amount of CPU resources during application execution. On the other hand, the changes of a device's contact surface and orientation can impact the heat transfer coefficient of the device and affect the system-on-a-chip (SoC) temperature. Thus, energy-efficient continuous sensing and recognizing users' activities and environment is necessary for modern power-limited mobile devices (Section 3.4). Table 3 classifies the energy optimization techniques for mobile devices in a more granular level.

3.1 Design Operating Systems

Roy et al. [2011] argued that modern mobile OSs need to provide isolation, delegation, and subdivision, which are three key mechanisms in an OS for energy usage. However, few existing works

Table 3. A Fine-grained Classification of Energy Optimizations on Mobile Devices

Classification	References
Mobile OS	[Chu et al. 2011] [Roy et al. 2011] [Rodriguez and Crowcroft 2011]
Display	[Chu et al. 2011] [Maghazeh et al. 2015] [Kim et al. 2014] [Lee et al. 2014]
Memory	[Chu et al. 2011] [Roy et al. 2011] [Hsieh et al. 2015] [Lee et al. 2014]
PDN	[Lee et al. 2014]
Sensors (e.g., GPS and accelerometer)	[Chu et al. 2011] [Liang et al. 2014] [Paek et al. 2010] [Lin et al. 2010] [Kjærgaard et al. 2011] [Priyantha et al. 2011] [Lee et al. 2014] [Liu et al. 2013b] [Yan et al. 2012]
Processors (e.g., CPU, GPU, and application processor)	[Tian et al. 2018] [Chu et al. 2011] [Roy et al. 2011] [Rodriguez and Crowcroft 2011] [Priyantha et al. 2011] [Maghazeh et al. 2015] [Hsieh et al. 2015] [Lee et al. 2014] [Singla et al. 2015] [Pathania et al. 2015] [Pathania et al. 2014] [Chuang et al. 2017] [Kadjo et al. 2015] [Kim et al. 2015] [Li et al. 2013c] [Chen et al. 2015] [Yang et al. 2013a] [Paterna and Rosing 2015]
Prediction algorithms	[Chu et al. 2011] [Maghazeh et al. 2015] [Hsieh et al. 2015] [Singla et al. 2015] [Chuang et al. 2017] [Kim et al. 2015] [Nirjon et al. 2012] [Donohoo et al. 2012] [Deng and Balakrishnan 2012] [Bui et al. 2013] [Li et al. 2013a] [Tong and Gao 2016] [Habak et al. 2015] [Gai et al. 2016]
Green mobile application or service	[Kjærgaard et al. 2011] [Maghazeh et al. 2015] [Chuang et al. 2017] [Han et al. 2016b] [Li and Halfond 2014] [Cun et al. 1990] [He et al. 2017]
User activity and environment	[Liang et al. 2014] [Kim et al. 2015] [Yan et al. 2012] [Li et al. 2013c] [Chen et al. 2015] [Yang et al. 2013a] [Paterna and Rosing 2015] [Rodriguez and Crowcroft 2011]
Energy bugs	[Jiang et al. 2017] [Pathak et al. 2011] [Pathak et al. 2012] [Liu et al. 2013b]

have comprehensively studied these three aspects. They designed a new OS, Cinder, to better understand and control energy use for energy-constrained mobile devices. In Cinder, two new abstraction concepts are presented to support the above three mechanisms for controlling the energy use of applications. These two new concepts are called reserves and taps, which are responsible for storing and distributing energy for the use of applications, respectively. Reserves are in charge of fine-grained energy accounting and allocating a given amount of energy to applications. Taps determine and control the energy flowing rate limits between these reserves. The collaboration between reserves and taps gives an opportunity for OSs to manage energy based on intentions of applications. Experimental results show that compared with uncooperative radio access, their cooperative resource sharing method in Cinder can save 12.5% power for mobile devices.

Mobile hardware and traditional OS vendors have developed different approaches (e.g., isolation, delegation, and subdivision) to extending the device battery life at various levels. However, their efforts are limited by the strict stratification of OSs, which makes it difficult to achieve cross-layer optimizations. Rodriguez et al. [2011] observed that the resource states of mobile devices and the device usage patterns and habits of users are different; thus, they presented a user-centered and energy-aware mobile OS, ErdOS, on top of Android OS to save energy for mobile devices. ErdOS adopts two techniques and integrates them together to reduce energy consumption. The first

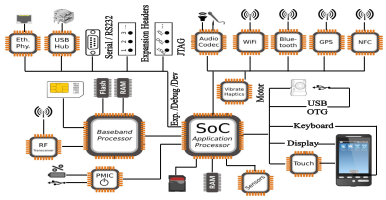


Fig. 3. Mobile hardware architecture [Yaghmour 2012].

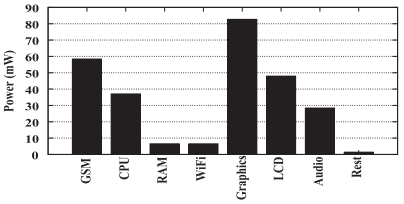
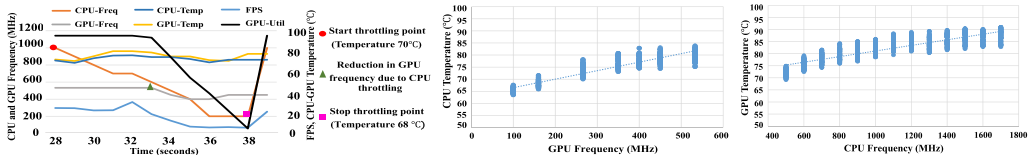


Fig. 4. Hardware power consumption [Carroll and Heiser 2010].



(a) Effect of thermal throttling on frequency. (b) Increase in CPU temperature. (c) Increase in GPU temperature.

Fig. 5. Effect of thermal throttling on frequency and temperature in Linux governor [A. Prakash and Henkel 2016].

technique is essentially a proactive resources management system that uses users' habits and customs to predict their future resource requirements of mobile applications to achieve better resources management. In particular, ErdOS adopts contextual information-based clustering algorithms to learn and infer users' activities. The second technique leverages low-power Bluetooth technology and inter-process communication to explore opportunistic access to neighboring devices' available resources to provide access control strategies. Experimental results show that ErdOS can save 11% energy as compared to the case in which each device accesses the GPS receiver locally.

3.2 Optimize Hardware Modules

Mobile devices usually consist of processors (e.g., CPU, GPU, and application processor), memory, sensors (e.g., accelerometer, compass, and GPS), display, camera, and wireless network interfaces. Figure 3 shows a general mobile hardware components architecture [Yaghmour 2012]. These components are energy consumers and contribute different levels of power consumption to the exhaustion of the battery. Figure 4 shows the average power consumption of partial hardware components in mobile devices [Carroll and Heiser 2010]. Usually, the power consumption of a component consists of static and dynamic power consumption [Xiao 2011]. The former relies on the hardware physical characteristics while the latter is decided by the applications or services running on OSs. It is crucial to understand the relationship among device power consumption, hardware activities, and software operations. Much of the research has optimized the energy of mobile devices by optimizing these power-hungry hardware modules. Next, we introduce these research works in detail.

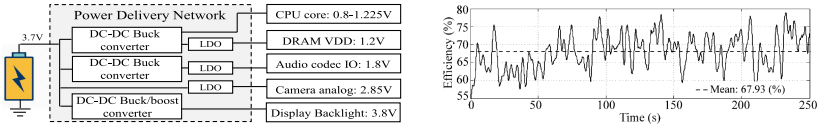
Since the CPUs and GPUs on heterogeneous MPSoCs are very close, even a moderate increase in temperature will directly affect the temperature variation of other components. We call this phenomenon as *thermal coupling effect* [A. Prakash and Henkel 2016]. Figure 5 studies the impact of thermal throttling on the frequency in a Linux governor and shows the relationship between temperature and frequency of processors. From Figure 5(a), we can see that due to thermal management imbalance between CPUs and GPUs and thermal inertia, the MPSoC temperature continues to increase to beyond 70°C, causing a significant processor frequency throttling for reducing chip

temperature. From Figure 5(c) and (c), we can see that when the frequency of the GPU (CPU) increases, the CPU (GPU) temperature also rises. Usually, hotter devices require more power to maintain normal operations. Thus, it is necessary to perform cooperative thermal management for mobile devices with multiple processors (e.g., CPU and GPU).

Pathania et al. [2014] noted that 3D mobile games, which require collaboration between CPU and GPU on SoCs, usually consume significant energy. In order to achieve efficient energy management for mobile devices, they presented a CPU-GPU power management mechanism by performing dynamic voltage and frequency scaling (DVFS) dynamically while maintaining stable performance. This mechanism achieves the range of target frames per second (FPS) with minimal power by dynamically exploiting CPU-GPU DVFS capabilities. However, the method in this work assumes that the relationship between FPS and CPU-GPU costs is perfectly linear, thus leading to power inefficiency. Pathania et al. [2015] overcame this shortcoming by developing models to capture the complex dynamics involved in the relationship. They used linear regression models to study the relationship between power and performance under a wide range of operating conditions. Meanwhile, they integrated an online learning method to further refine the parameters at runtime to make the models more accurate. Based on the models, a power management strategy is proposed to efficiently optimize device power consumption. Experimental results show that compared with default Linux power managers [Pallipadi and Starikovskiy 2006] and an integrated power manager [Pathania et al. 2014], their proposed method can achieve 29% and 20% higher power-efficiency (i.e., FPS/Watt) for mobile games.

Chuang et al. [2017] adopted power management methods in Pathania et al. [2015] and developed an online learning method on top of it. The method learns the relationship between energy consumption and performance at runtime based on historical data to save energy. Experimental results show that their governor saves 26% device energy as compared to Linux performance governor. Kadjo et al. [2015] also used FPS to characterize user experience. They argued that different applications require different computing resources to satisfy the desired FPS of applications. They optimized energy consumption of graphic applications by managing CPU and GPU frequencies dynamically. The interplay between CPU and GPU is first modeled as a queuing system, and an energy-aware coordination method is then used to alter CPU's and GPU's frequencies. Specifically, CPU generates commands and inserts them into a queue while GPU executes the commands ejected from the queue. Further, a controller is formulated to adjust the frequencies of CPU and GPU dynamically for energy optimization. Experimental results show that compared with Android power management, their method can save 17.4% energy for mobile devices.

Paek et al. [2010] noted that GPS is more accurate than other positioning systems but also more power-hungry. In urban areas, GPS is usually imprecise; thus, it is reasonable to compromise position accuracy to save energy. They designed an adaptive positioning mechanism called Rate-Adaptive Positioning System (RAPS) that combines novel techniques to decide when to activate GPS. First, RAPS leverages a duty-cycled accelerometer to detect users' movement. Second, RAPS stores the detected user location information to turn on GPS adaptively. Third, RAPS detects GPS unavailability (e.g., indoors) by employing celltower-RSS blacklisting and avoids activating GPS in these places. Lin et al. [2010] made full use of the available sensing mechanisms in devices to enable them to work in tandem to avoid unnecessary energy waste. They presented a-Loc, a system location service, to adaptively manage location sensors, accuracy, and device energy. a-Loc adopts Bayesian estimation to capture and construct users' location, and decides applications' accuracy requirements dynamically. Based on accuracy requirements, a-Loc adaptively adjusts energy expenditure using the available sensors. However, Kjærgaard et al. [2011] noted that many applications require not only the current position but also the historical motion. Thus, they presented a sensor management strategy to sense trajectory and decide when to sample data for



(a) The PDN in a smartphone platform. (b) Power conversion efficiency of PDN.

Fig. 6. The conceptual diagram and the power conversion efficiency of the PDN in a smartphone platform [Lee et al. 2014].

different sensors (e.g., accelerometer, compass, and GPS). Further, they proposed multiple trajectory updating protocols to cooperatively save power using trajectory simplification algorithms to simplify data before sending it to remote servers. Finally, they designed an energy-efficient system that combines sensor management strategy with trajectory update protocols to optimize energy consumption of location-sensing and data transmission. Experimental results show that compared with the system of Lange et al. [2009], their method can save 5% device power.

Kim et al. [2014] noted that no change in display content can still result in frequent update frames of applications, thus wasting significant power. They designed an energy-effective strategy to save display energy without compromising graphical quality. They first defined content rate as the metric to represent the number of meaningful frames. Content rate decides a proper refresh rate for displaying content on display panels. Then, by using double buffering and grid-based comparison, a runtime system is designed to measure the content rate dynamically. Finally, based on the changes in content rate, the methods of section-based control and touch boosting are developed to dynamically alter refresh rate. Specifically, the former method implements a match between the content rate and the refresh rate based on a predefined section table. This method cannot respond to the sudden changes of the frame rate. Thus, the latter boosting technique is further proposed to enforce the refresh rate when a touch event occurs. Experimental results show that their method can reduce 230 mW power consumption compared with devices that do not use their method. Maghazeh et al. [2015] observed that significant power can be saved by scaling resolution and users are willing to accept a lower resolution when battery power is low. Thus, they presented a perception-aware adaptive method to scale resolution when playing a game. The method uses a powerful statistical model to quantify user perceptions and predict user perceptions dynamically based on this model. Further, an online heuristic is presented to adaptively adjust resolution based on system dynamics to maintain the tradeoff between system power and user perception of the quality. Experimental results show that the method can save 70% power as compared with a default resolution (i.e., 3860×2160).

Hsieh et al. [2015] explored the impact of the memory access footprint on energy consumption of mobile games. They designed a memory-aware scheme to optimize energy by considering CPU, GPU, and memory governors cooperatively. They first analyzed memory usage characteristics of off-chip during game play, and found that the time it takes to access main memory is critical to graphics performance. Based on this observation, they constructed a regression model to forecast memory accesses time during frame rendering. Finally, based on the predicted time taken to access memory, a cooperative memory-aware CPU-GPU governor is proposed to make a DVFS decision in case of pushing out a new frame. Experimental results show that compared to default system governors, their method can save 13% device energy.

The above works ignore the power conversion efficiency from the battery to other modules. As shown in Figure 6(a), the power delivery network (PDN) of a mobile device consists of multiple dc-dc converters, in which the power dissipations can lead to massive power loss. Figure 6(b) shows that the overall device power efficiency is about 70%, meaning that a significant energy will

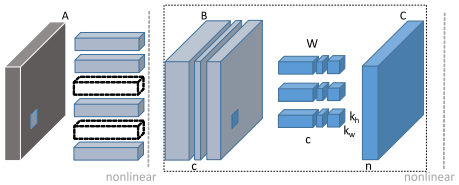


Fig. 7. Convolutional layer channel pruning [He et al. 2017].

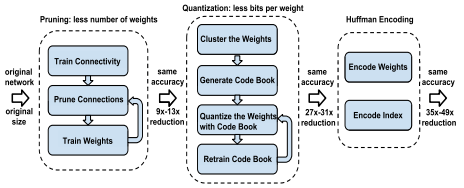


Fig. 8. Three-stage compression pipeline [Han et al. 2016b].

be wasted when the power converts from the battery to other modules. Lee et al. [2014] studied the relationship between power converter configurations and various operating conditions. They found that the efficiency of DC-DC converters drops dramatically when DC-DC converters are operating under undesirable operating conditions. Thus, they designed DC-DC converter models for each module in mobile devices and leveraged linear regression to analyze power conversion efficiency. Two optimization methods are proposed to minimize power loss of DC-DC converters. One is a static switch method that maximizes the efficiency of DC-DC converters based on statistical load behavior information. The other is a dynamic switch method that achieves high efficiency enhancement by adaptively controlling DC-DC converter switches under varying load conditions. Experimental results show that their method can reduce 19% energy consumption for mobile devices.

3.3 Improve Mobile Applications and Services

The improvements of hardware and battery techniques cannot stop poorly designed applications from unnecessarily draining the battery power. Existing tools (e.g., cycle-accurate simulators [Brooks et al. 2000], power monitors [McIntire et al. 2006], program analyses [Hao et al. 2013], and statistical-based measurement techniques [Li et al. 2013b]) could help developers understand application energy usage patterns. However, they do not provide guidelines about how to modify code for energy reduction. Li et al. [2014] conducted extensive evaluations of commonly used application programming practices to provide guidance for application developers. In particular, they collected a flock of developer-oriented tips and recommendations from the official Android developers website. Among these recommended practices, they mainly studied network usage, memory consumption, and low-level code practices, which are rigorously evaluated to judge whether they are useful for application developers. In particular, for network usage, they found that when investigating the code practices of making HTTP requests, transmitting larger files is more energy-efficient as compared with transmitting smaller files. For memory consumption, they found that high memory usage usually consumes a small amount of energy of applications. For low-level programming practices (e.g., static invocations), their experimental results show that static invocations can reduce 15% energy consumption as compared with virtual invocations. The above investigative results offer valuable guidance to application developers to optimize the energy consumption of mobile applications.

Developers can benefit from code practices to design energy-efficient applications. However, this approach may be inefficient for deep neural network (DNN)-based applications (e.g., object recognition) that require large computation and storage overheads of devices. Thus, reducing neural network size is an efficient method to decrease computation and storage requirements when deploying DNNs on devices. Cun et al. [1990] proposed a method, Optimal Brain Damage (OBD), to reduce learning network size by selectively deleting weights. To this end, a theoretically justified measure method is proposed to compute weight saliency. The weight parameters with small

"saliency" can be deleted since their deletions hardly affect the training error. Experimental results show that OBD can reduce the amount of neural network parameters by four times. Different from selectively deleting weights, He et al. [2017] introduced a novel channel pruning scheme for compacting the convolutional neural network (CNN) by utilizing the redundancy inter-channels. Figure 7 shows the process of channel pruning for a single convolutional layer. The goal of the channel pruning method in this figure is not only to reduce the feature map B's width but also to maintain the feature map C's outputs. If the channels have been trimmed, the corresponding filters' channels having these trimmed channels as inputs can be removed. In addition, the filters that generate these channels can also be deleted. Usually, channel pruning involves channel selection and reconstruction. Motivated by this, they further proposed an iterative algorithm including two stages. First, they used LASSO regression to select the most typical channels and remove the redundant channels. Then, they adopted the linear least squares method to reconstruct the outputs based on the remaining channels. Experimental results show that their method can achieve 2x acceleration for ResNet while suffering only 1.4% accuracy loss.

Han et al. [2015] introduced a novel approach to reduce network size by removing superfluous connections to reduce the number of weights. On top of this method, they proposed a deep compression method, which contains two crucial steps including trained quantization and Huffman coding, to further compress deep neural networks, as shown in Figure 8 [Han et al. 2016b]. Trained quantization aims to quantize the pruned weights to make multiple connections that share the same weight. Huffman coding as a lossless data compression technique is utilized to encode the weights by using the variable-length codewords to ease the pressure on network storage. Compared with the original network, their proposed pruned network layer takes 7x less energy on CPU, 3.3x less on GPU, and 4.2x less on mobile GPU on average. Based on the method of deep compression, an efficient inference hardware accelerator, Efficient Inference Engine (EIE), is further designed to work on this compressed model [Han et al. 2016a]. Experimental results show that compared with DaDianNao (Application-Specific Integrated Circuit (ASIC)), EIE can achieve 2.9x better throughput, 19x energy efficiency, and 3x area efficiency.

3.4 Sense Activities and Environments

The above works ignore the interactions between devices and users when optimizing energy consumption. In fact, it turns out that fully exploring behaviors and activities of mobile users can help save device energy. For example, ErdOS [Rodriguez and Crowcroft 2011], which has been introduced in Section 3.1, is a user-centered and energy-aware mobile OS. In ErdOS, two techniques are adopted and integrated to learn and infer users' activities. One is a proactive resources management system that infers users' habits and customs based on contextual information-based clustering algorithms. The other explores the opportunistic access to neighboring devices' available resources using connections between users to provide access control strategies. Experimental results show that compared with accessing the GPS receiver locally, ErdOS can save up to 11% energy by learning users' activities and sharing GPS reads with nearby devices.

The following works optimize device power consumption based on user experience and user satisfaction. Li et al. [2013c] investigated the correlations among user experience, runtime system activities, and the minimal required frequency of an application processor. Based on these correlations, they designed an inference model that inputs the runtime statistics and outputs the most energy-efficient power state of the application processor. Using this inference model, an adaptive power management scheme, SmartCap, is proposed to automatically identify the most energy-efficient state of the application processor according to the system activities. Chen et al. [2015] put forward a user experience-oriented CPU-GPU governing framework to save energy without posing significant impact on user experience. This framework includes three modules. The first

module monitors application executions and identifies users' demands at runtime. Based on the monitoring results, the second module adjusts the policies for processors' governors dynamically. To prevent the module from casually making frequency-scaling decisions for processors, the third module is responsible for communicating between two modules to interpret the frequency-scaling intents. Yang et al. [2013a] explored the effect of CPU utilization on user satisfaction. They found more than a simple linear relationship between user satisfaction and the CPU utilization. They proposed an adaptive CPU frequency scaling scheme, Human and Application driven frequency scaling for Processor Power Efficiency (HAPPE), which takes the users' and applications' requirements into account. The scheme learns the necessary CPU frequencies satisfying the requirements of users and applications for each CPU utilization level using a learning algorithm. The learning algorithm contains two periods. The first period is the training period. For each user and application, the algorithm trains the user-application frequency profile based on users' inputs the first time running this application. Specifically, a few keystrokes are used as the users' inputs to explore their satisfactory CPU frequency level. Then, the desired CPU frequencies at different CPU utilization levels are stored. After the training period, users do not need to provide the inputs unless their requirements change. If the requirements change, HAPPE will learn the new necessary CPU frequencies to satisfy the new requirements. Experimental results show that their methods can save 45.1%, 84%, and 25% energy as compared with the governor in Pathania et al. [2014], Linux ondemand DFS strategy, and default Linux CPU frequency controller, respectively.

Kim et al. [2015] observed that users' behaviors (i.e., zooming and scrolling) usually demand a different amount of CPU resources during application execution. They designed a user activity recognition framework to sense user's activities at runtime and make system decisions to reduce power consumption. Three crucial modules are contained in this framework. The first module makes activity profiles for each user's interaction, which are processed by event handlers. The event handlers are the functions that solve diverse user interactions. Based on a user's activity profile, the second module constructs the user activity model for each application to classify the obtained activity profiles. According to the user activity model, the third module recognizes the user's activities dynamically at runtime. Finally, an energy-efficient power and thermal management method is designed to adjust the system configurations according to the user's activities dynamically. Experimental results show that compared to a default Linux ondemand governor, their proposed framework can achieve 28% reduction in CPU energy consumption.

The following two works investigate energy-efficient device management for continuous user activity recognition (e.g., sit, stand, or walk). Yan et al. [2012] learned the characteristics of the energy consumed by the activity recognition applications and found that two independent parameters, sampling frequency and classification features, can impact the energy consumed by applications. Different configurations of these two parameters are required by different activities to realize their diverse accuracy levels. Based on this observation, an adaptive activity recognition mechanism is presented to recognize the activities continuously, and adjust two parameters dynamically for these activities. Liang et al. [2014] found that it is reasonable to shorten the operating time of the activity recognition sensors through reducing sampling frequency. They designed a hierarchical recognition approach to recognizing the users' activities by setting low sampling frequency and reducing frequency-domain feature extraction to achieve long-term activity monitoring. Like Kim et al. [2015], this approach first models the templates for each activity by extracting features from the sampled data. Then, the features are extracted from the sliding window and the similarities between the target activity and the templates are calculated. Based on the similarities, the target activity is classified into the corresponding template type. Experimental results show that their methods can save 25% power as compared with a non-adaptive recognition scenario, and can extend battery life by 3.2 hours by setting a low sampling frequency, respectively.

Different from the above works that concentrate more on mobile users, Paterna et al. Paterna and Rosing [2015] investigated the influence of the ambient environment on the energy consumed by devices. They observed that the changes of device's contact surface and orientation can impact the heat transfer coefficient of the device and affect the SoC temperature. Thus, they addressed the heat transfer variations and the thermal coupling effect among the components on mobile devices. First, under different operating environment conditions, they analyzed the influences of the above ambient variations on the thermal profile. Subsequently, a variation-aware strategy is presented to model the thermal behavior by using the data from available corresponding sensors. Finally, a proactive thermal control management method is devised to manage both the frequencies of CPU and GPU to adapt to the changes of users and environments while meeting the temperature constraints. Experimental results show that compared with existing techniques that consume as much as $1W$ on average, their proposed method consumes $0.6W$ power on average.

3.5 Summary

Shannon's law predicts that the transmission performance can be doubled in 8.5 months, the battery manufacturers need to spend at least 5 years to achieve similar growth in power density, and the memory performance takes 12 years to achieve 2x improvement. The development speed of hardware technology falls behind the growth rate of application requirements. Usually, high performance demands more power. Hence, it is crucial to trade off performance and power within a particular design. In this section, we summarize energy optimization methods from perspectives of mobile OSs, mobile hardware, mobile application or services, user activities, and ambient environment perception. Current mobile devices face a new class of abnormal system behaviors that could happen in any entities in a device (e.g., hardware, application, or OS), leading to serious battery drain [Pathak et al. 2011]. Effective approaches for diagnosing and handling these energy bugs could be found in Jiang et al. [2017] and Liu et al. [2013b]. Modern mobile OSs are modified based on a general-purpose OS without regarding energy efficiency as a goal. In fact, mobile OSs need not only the abilities to provide energy usage patterns for components and applications, but also the capacities to adaptively control and manage system energy and resource usages for applications. For mobile hardware, energy-efficient and component-specific algorithm developments and low-power component designs can be utilized to save energy at multiple levels (e.g., transistor-level). For example, Kumar et al. [2010] argued that adopting a new generation of semiconductor technology can reduce device energy consumption. However, given Moore's law, the semiconductor manufacturers need to spend 1.5 years to improve the microprocessor performance by two times. As transistors become smaller, even though each transistor consumes less power, more transistors for better performance will cause more power consumption.

DVFS can efficiently manage device power by adjusting the processor clock frequency and the supply voltage. It is widely used in computer hardware to maximize power savings while still maintaining ready compute performance availability. Hence, combining DVFS and efficient prediction algorithms could provide devices more energy-saving opportunities by proactive managing system resources. The design principle of most mobile applications running on devices are usability factors-driven rather than energy efficiency-driven. Thus, application-oriented energy optimization methods are necessary to provide more opportunities for further energy reduction, especially for computation/storage/communication-intensive applications. The convergence of MC and CC has spawned a new computing paragram, namely MCC. In the context of MCC, the data storage and processing run outside of mobile devices. In this way, the restrictions of mobile devices such as battery life and storage capacity can be effectively solved. In the next section, an introduction on computation offloading strategies for these computation/storage/communication-intensive applications is given in detail. Computation offloading can alleviate the device energy pressure by

Table 4. A Comparison Summary of Device Energy Optimization Methods from Multiple Aspects, Such as Optimization Target, Power/Energy Savings, Performance Improvement, Benchmark Schemes, and Experimental/Simulation Platforms

References	Optimization target	Power/Energy	Performance	Benchmarks	Platforms
[Roy et al. 2011]	OS	+12.5%	*	non-coop	HTC Dream
[Yang et al. 2013a]	CPU	+25%	*	Linux controller	Linux laptop
[Kim et al. 2015]	CPU	+9.5%	*	[Li et al. 2013c]	MDP8660
[Pathania et al. 2015]	CPU-GPU	+29%	+29%	Linux power manager	Odroid-XU+E
[Chuang et al. 2017]	CPU-GPU	+26%	*	Linux performance governor	Google Nexus 7 device
[Kadjo et al. 2015]	CPU-GPU	+17.4%	−0.9%	Android power management	Intel Baytrail-based Android
[Singla et al. 2015]	CPU-GPU	+16%	*	use a fan	Linux kernel
[Chen et al. 2015]	CPU-GPU	+45.1%	*	[Pathania et al. 2014]	Samsung Galaxy S4
[Paterna and Rosing 2015]	CPU-GPU	+40%	*	[Rodero and et al. 2010]	MSM8660
[Li et al. 2013c]	app processor	+11%∼+84%	*	linux DFS strategy	Motorola ME525
[Paek et al. 2010]	GPS	+3.8x	*	always on GPS	Nokia N95
[Lin et al. 2010]	GPS	+45%	*	periodic GPS	Android G1
[Kjærgaard et al. 2011]	GPS	+36%	*	EnTracked	Nokia N97 device
[Priyantha et al. 2011]	processor & sensor	+3 orders of magnitude	*	current HTC Touch Pro architecture	HTC Touch Pro
[Yan et al. 2012]	accelerometer	+20%	*	non-adaptive recognition scenario	Samsung Galaxy S2
[Liang et al. 2014]	accelerometer	+3.2 hours	*	sampling frequency (20Hz)	Samsung i909
[Kim et al. 2014]	display	+230 mW	−5%	default method	Galaxy S3 LTE
[Maghazeh et al. 2015]	display	+70%	*	default resolution (3860×2160)	Odroid-XU3 board
[Hsieh et al. 2015]	memory	+13%	*	default governors	Odroid-XU3
[Lee et al. 2014]	PDN	+19%	*	default method	MSM8660
[Li and Halfond 2014]	coding practices	+15%	*	virtual invocations	Samsung Galaxy II
[He et al. 2017]	CNN	*	+5x	tensor factorization	CNN
[Han et al. 2016a]	DNN	+19x	+2.9x	DaDianNao (ASIC)	DNN
[Pathak et al. 2012]	ebug	+20%∼+65%	*	default method	Android

* indicates that the corresponding reference does not provide specific performance discussions and analysis.

offloading computational parts of an application to cloud servers for execution. Table 4 gives a summary of the performance comparisons of device energy optimization techniques.

4 OFFLOAD COMPUTATION FROM DEVICES

The computing and storage resources, as well as severe energy constraints caused by limited battery life, are the major bottlenecks for mobile devices. Fortunately, offloading is an effective solution that enhances devices' computing and storage capabilities through sending computational tasks to powerful servers for execution. Most research on offloading decisions focuses on two areas: improving performance and saving energy. In this section, we mainly focus on offloading

strategies that target energy efficiency improvement for mobile devices. The process of offloading usually includes three crucial steps, that is, application partition, preparation, and offloading decision [Akherfi et al. 2018]. Application partition divides an application into the offloadable and the non-offloadable portions. The former portions (i.e., offloadable) will be migrated to the powerful servers (e.g., remote clouds or edge clouds) for execution while the latter portions (i.e., non-offloadable) will be executed locally. Preparation performs necessary actions for offloadable parts to enable their successful offloading and execution, including remote servers selection, code transmission, and the like. The offloading decision is the final step that makes offloading decisions for the offloadable parts before transmitting them to the remote cloud servers or the edge servers. In particular, based on the offloading object, offloading can be divided into data offloading and computing offloading. The former migrates data from one network to another while the latter migrates a computing process to the clouds for execution. Moreover, according to the offloading methods, offloading can be classified into fine-grained (or partial) offloading and coarse-grained (or complete) offloading. Fine-grained offloading strategies divide an application and only offload parts of the application and transmit as little data as possible, while coarse-grained offloading strategies usually migrate the entire application to a cloud server to ease the burden on the application programmers.

Many researchers have attempted to enable mobile devices to apply remote execution to improve energy efficiency and application performance. Since offloading migrates calculations to computers with richer resources, it involves whether to migrate calculations and which calculation decisions to migrate. Computation offloading is productive only when energy savings from computation offloading exceed the cost incurred by other factors (e.g., data communication, privacy and security guarantee) (Section 4.1). Once determined to perform computation offloading, the application offloading decision should be judiciously made before sending the offloaded portions to clouds (Section 4.2). It is noted that if the processes of partition, preparation, and offloading decisions are all done at the design process of an application, then this offloading mechanism is called static offloading, which does not consider the real execution context, thus may be energy-inefficient. Motivated by the highly dynamic characteristics of wireless network channels, dynamic offloading is emerging to make offloading decisions at runtime to adapt to different network conditions (Section 4.3). In this context, an application is partitioned with different granularity levels and the offloadable parts are dynamically migrated to powerful servers to adapt to the unstable network environment. Table 5 summarizes the energy-efficient offloading strategies for computation/storage/communication-intensive mobile applications.

4.1 To Offload or Not to Offload?

Mobile devices can benefit from computation offloading to extend the battery lifetime and improve user experience. However, Nguyen et al. [2016] argued that it is crucial to explore the energy usage of local execution and network communication before performing offloading. The following works discuss the factors that need to be considered before performing computation offloading. Namboodiri et al. [2012] analyzed the energy characteristics for the cloud and the non-cloud versions of an application to judge which of them can save more energy in MCC. First, they compared the energy consumed by three mobile applications (i.e., word processing, multimedia, and games) under cloud and non-cloud versions on mobile devices with different form-factors. Second, an analytical model is proposed to generalize these comparison results and adaptively analyze the energy consumption of two application versions under dynamic communication and device processing capabilities. The model uses the metric, AppScore, to evaluate the performance of locally or remotely executed applications. Finally, based on the AppScore rating of two versions of the application, an algorithm called GreenSpot is developed to dynamically make the decision on which version is more energy-efficient. Specifically, it first accepts the users' request of the application

Table 5. A Classification of Energy-efficient Offloading Strategies

Classification	References
Security	[Roy et al. 2011] [Rodriguez and Crowcroft 2011] [Kumar and Lu 2010] [Mtibaa et al. 2014]
Communication	[Namboodiri and Ghose 2012] [Guo and Liu 2018] [Shu et al. 2013] [Chun et al. 2011] [Barbera et al. 2013] [Cuervo et al. 2010] [Kovachev et al. 2012] [Lin et al. 2014] [Liu et al. 2016] [Qian and Andresen 2014] [Yang et al. 2013b] [Kwon and Tilevich 2013] [Zhang et al. 2017a] [Barbarossa et al. 2013] [Ding et al. 2013a] [Nirjon et al. 2012] [Dogar et al. 2010] [Hoque et al. 2014] [Bui et al. 2013] [Li et al. 2013a] [Zhang et al. 2013] [Zhang et al. 2015a] [Tong and Gao 2016] [Yao et al. 2013] [Zhang et al. 2015b] [Kosta et al. 2013] [Terefe et al. 2016] [Goudarzi et al. 2017] [Zhang et al. 2017b] [Zhang et al. 2018] [Shi et al. 2012] [Magurawalage et al. 2014] [Gai et al. 2016] [Zhou et al. 2015] [Rahimi et al. 2012] [Shi et al. 2014]
Coarse granularity	[Namboodiri and Ghose 2012] [Shiraz and Gani 2014] [Kosta et al. 2012] [Barbera et al. 2013] [Cuervo et al. 2010] [Xia et al. 2014] [Li et al. 2013a] [Zhang et al. 2013] [Zhang et al. 2015a] [Kosta et al. 2013]
Fine granularity	[Guo and Liu 2018] [Chun et al. 2011] [Kovachev et al. 2012] [Lin et al. 2014] [Liu et al. 2016] [Balakrishnan and Tham 2013] [Kemp et al. 2010] [Qian and Andresen 2014] [Xia et al. 2014] [Yang et al. 2013b] [Kwon and Tilevich 2013] [Zhang et al. 2017a] [Barbarossa et al. 2013] [Tong and Gao 2016] [Yao et al. 2013] [Zhang et al. 2015b] [Kwon and Tilevic 2012] [Wang et al. 2013] [Terefe et al. 2016] [Goudarzi et al. 2017] [Fan et al. 2017] [Zhang et al. 2017b] [Zhang et al. 2018] [Chen et al. 2018] [Shi et al. 2012] [Mtibaa et al. 2014] [Habak et al. 2015] [Wu et al. 2016] [Magurawalage et al. 2014] [Gai et al. 2016] [Ravi and Peddoju 2015] [Zhou et al. 2015] [Rahimi et al. 2012] [Shi et al. 2014]
Static	[Li et al. 2001] [Xian et al. 2007] [Xia et al. 2014]
Dynamic	[Guo and Liu 2018] [Wu and Wolter 2015] [Lin et al. 2013] [Shiraz and Gani 2014] [Ge et al. 2012] [Shu et al. 2013] [Chun et al. 2011] [Kosta et al. 2012] [Cuervo et al. 2010] [Kovachev et al. 2012] [Lin et al. 2014] [Liu et al. 2016] [Balakrishnan and Tham 2013] [Kemp et al. 2010] [Qian and Andresen 2014] [Yang et al. 2013b] [Kwon and Tilevich 2013] [Zhang et al. 2017a] [Barbarossa et al. 2013] [Ding et al. 2013a] [Li et al. 2013a] [Zhang et al. 2013] [Zhang et al. 2015a] [Tong and Gao 2016] [Yao et al. 2013] [Zhang et al. 2015b] [Kwon and Tilevic 2012] [Kosta et al. 2013] [Wang et al. 2013] [Terefe et al. 2016] [Goudarzi et al. 2017] [Beck et al. 2015] [Fan et al. 2017] [Zhang et al. 2017b] [Zhang et al. 2018] [Chen et al. 2018] [Shi et al. 2012] [Mtibaa et al. 2014] [Habak et al. 2015] [Wu et al. 2016] [Magurawalage et al. 2014] [Gai et al. 2016] [Ravi and Peddoju 2015] [Zhou et al. 2015] [Rahimi et al. 2012]

as the input; then, it judges which version of this application is available and determines the most energy-efficient version based on the analytical results on energy consumption.

Further, Kumar et al. [2010] considered the data privacy and security issues during offloading. To prevent data from being attacked, they argued that the offloaded data needs to be preprocessed (e.g., data encryption) before sending to the clouds. However, these operations will inevitably incur additional energy consumption for mobile devices, which has a very important impact on the offloading decision. Barbera et al. [2013] were concerned more about the significant communication cost between mobile devices and clouds, especially under poor network connectivity. They investigated the feasibility of computation offloading and data backups between mobile devices and clouds. In this context, each mobile device has two types of software clones in the cloud, namely, off-clone and back-clone. The former is used to perform computation offloading while the latter is used to restore data. Based on the analysis of network availability and signal quality, they evaluated and reported the necessary communication cost to achieve synchronization between devices and clones. Finally, they presented Logger, an Android application to continuously record

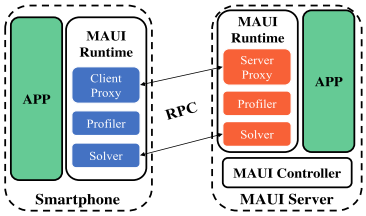


Fig. 9. The high-level view of MAUI [Cuervo et al. 2010].

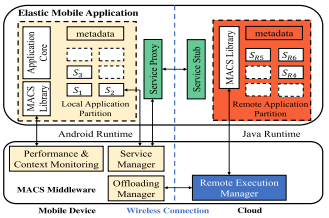


Fig. 10. The architecture of MACS [Kovachev et al. 2012].

and collect the events occurring in the device to study the overheads incurred by the two clones. Measurements results show that Logger can accurately assess the feasibility and communication costs of two clones. In general, before performing offloading, factors such as data privacy and security, wireless communication overhead, and cloud availability need to be carefully considered and evaluated to achieve green offloading.

4.2 Partition and Prepare Offloadable Tasks

Cuervo et al. [2010] designed a fine-grained offloading system, MAUI, as shown in Figure 9, that not only performs fine-grained partition but also reduces the burden on programmers. With MAUI, the developers only need to provider an initial partition of applications. In particular, MAUI leverages several properties of the managed code environment to achieve the goal. First, *code portability* is used to make two versions of an application. More specifically, one version is processed locally while the other version is processed remotely. Subsequently, MAUI combines *reflection* and *type safety* to determine the offloadable parts of an application dynamically. Further, MAUI evaluates each offloadable part and uses *serialization* to decide its network transmission overhead. Finally, based on the measurements of transmission and computing overheads, the offloading process is formulated as a linear programming problem, and the solution can obtain the network condition-aware dynamic application partition methods to minimize device energy consumption. Experimental results show that MAUI can achieve about 10x, 27%, and 45% energy savings for face recognition, video game, and chess, respectively.

Inspired by MAUI [Cuervo et al. 2010], Kovachev et al. [2012] developed an effective offloading strategy to partition and offload computational tasks of an application to the cloud at runtime to reduce device energy consumption. They designed Mobile Augmentation Cloud Services (MACS), as shown in Figure 10, a middleware to enable adaptive partition and offloading between mobile devices and clouds. First, MACS develops an offloadable lightweight application model to indicate the computational tasks corresponding to the services encapsulating specific functionalities. Then, based on available devices and communication conditions (e.g., CPU load and bandwidth), the partition decision is formulated as an optimization problem to divide the application into two parts. Finally, according to the optimization solution, MACS offloads the corresponding portions of the application to the cloud for execution. Evaluation results show that MACS can achieve 95% energy savings as compared to local execution.

Lin et al. [2014] observed that task-level offloading can provide lower power consumption for mobile devices. They studied task scheduling and offloading problems in MCC to save energy under application completion time constraints. An application is divided into smaller tasks with execution requirements. They first proposed a scheduling algorithm to allocate the tasks within the minimal latency. Then, a task migration algorithm is designed to perform tasks offloading among local device cores and the remote cloud to achieve energy savings. Further, DVFS is utilized to

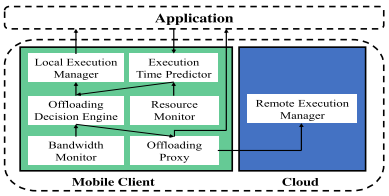


Fig. 11. The architecture of Phone-2Cloud [Xia et al. 2014].

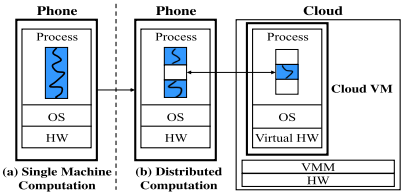


Fig. 12. The system model of Clone-Cloud [Chun et al. 2011].

adjust the frequencies of the local cores to further reduce the energy consumption. Finally, a linear-time rescheduling mechanism is presented to migrate the tasks to reduce the time complexity. Experimental results show that compared with local execution, their proposed method can achieve up to 74.9% energy savings.

Liu et al. [2016] argued that it is necessary to offload appropriate but not all the tasks to the cloud to reduce additional transmission cost to further save energy for mobile devices. In their work, an application can be partitioned into the smaller tasks including not only sequential tasks but also parallel tasks. They presented an energy-efficient scheduling mechanism to migrate the appropriate tasks to the cloud to save energy under the execution constraints. Specifically, an application is first decomposed into two types of the smaller tasks, and their execution is mapped into a directed acyclic graph (DAG). Subsequently, based on the hybrid tasks, they formulated the task dispatching problem as a constrained shortest path problem. Finally, they handled this problem by adopting the classical Lagrangian relaxation-based aggregated cost method and obtained the approximate optimal solution. Experimental results show that their method can achieve 81.93% energy reduction as compared with the local strategy.

Balakrishnan et al. [2013] observed that an application can be partitioned into the interconnected entities to form a Task Interaction Graph (TIG). Each TIG consists of vertices and edges. The vertices indicate the computation cost of tasks while the edges indicate the transmission cost between tasks. They studied the problem of assigning tasks to the external resources (e.g., cloud servers) and applying the DVFS technique to save energy for mobile devices. To this end, the task offloading is first formulated as a quadratic assignment problem to achieve the allocations of "task-resource" and "resource-frequency," respectively. Subsequently, they dealt with the allocation optimization problem by designing a two-level genetic approach. The inner level adopts the DVFS technique to identify the appropriate operating frequencies for resources, while the outer level uses the proposed slack time-based scheduling method to assign tasks in a TIG to these resources. Experimental results show that their method can save 35% energy compared with local execution.

4.3 Migrate Offloadable Tasks

Static partitioning strategies [Li et al. 2001; Xian et al. 2007] partition applications in the development phase. Xia et al. [2014] developed an offloading system, Phone2Cloud, to perform offloading for devices. Phone2Cloud consists of seven crucial components, as shown in Figure 11. Based on the methods in Kumar and Lu [2010] and Liu et al. [2010], Phone2Cloud provides a modified offloading-decision-making mechanism. Given an application, Phone2Cloud makes offloading decisions as follows. First, it uses time predictor to predict the average application execution time based on the CPU workload monitored by the resource monitor. Then, it compares the predicted result and the user-defined execution threshold of the application. If the predicted time is greater than the threshold, then parts of the application will be migrated to the cloud servers. Otherwise, it compares the energy consumption for processing the application locally and remotely. If the

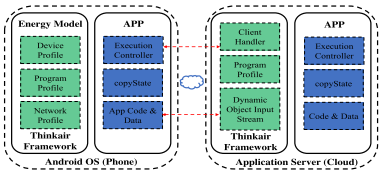


Fig. 13. The architecture of ThinkAir [Kosta et al. 2012].

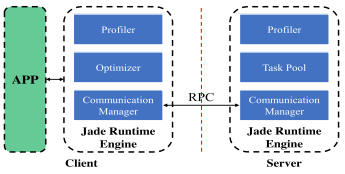


Fig. 14. High-level view of Jade [Qian and Andresen 2014].

remote energy consumption is greater than the local, then the entire application will be processed locally. Otherwise, parts of the application will be migrated to the cloud servers. Experimental results show that Phone2Cloud can save energy when the input size of "word count" is larger than 256KB.

Static offloading schemes cannot adapt to fluctuating network conditions, thus may be energy-inefficient in some cases. Kemp et al. [2010] presented a practical dynamic offloading mechanism, Cuckoo, to migrate the computational tasks to nearby mobile devices or the cloud servers. Cuckoo consists of a runtime system, a resource manager, and a programming model. The runtime system is configured to determine the target of maximizing computation speed or minimizing energy usage. The resource manager is designed for users to collect remote available resources for execution, such as laptops, servers, and other cloud resources. Considering the fact that mobile devices may be disconnected from the networks, the programming model is designed to provide an interface for developers to build applications supporting local and remote execution. Cuckoo simplifies the development of applications and provides a dynamic offloading system to determine the offloadable portions of an application at runtime. Experimental results show that Cuckoo can reduce the battery power consumption by 40 times compared with local execution.

Chun et al. [2011] argued that the partition methods for an application are different under different wireless network connections. They presented an offloading system, CloneCloud, as shown in Figure 12, to adaptively partition and send an application to the cloud device clone under dynamic wireless network. Specifically, given an application, like MAUI [Cuervo et al. 2010], CloneCloud first uses static analysis to identify the offloadable points of an application, then leverages dynamic profiling to evaluate the cost (e.g., energy consumption) of these possible offloadable points. Based on the cost model, a mathematical optimizer is adopted to select the offloadable points to perform offloading. Subsequently, the threads at the chosen points are offloaded from the mobile device to a cloud device clone (i.e., an application-level virtual machine). Finally, the threads executed on the device clone and their created new states at the same time are both returned to the device to combine with the initial process. Experimental results show that compared with local execution, CloneCloud can achieve up to 20x energy savings for mobile devices.

Kosta et al. [2012] solved the non-scalability of MAUI [Cuervo et al. 2010] and the limited environmental conditions considered of CloneCloud [Chun et al. 2011] by presenting a new offloading framework, ThinkAir, to achieve method-level offloading. As shown in Figure 13, first, ThinkAir uses the device clone in the cloud to process the offloadable methods to solve the non-scalability of MAUI. Second, ThinkAir adopts an online method-level offloading mechanism to overcome the restrictions in static analysis of CloneCloud by considering the hardware, software, and network conditions simultaneously. Finally, in the cloud side, ThinkAir performs on-demand resource allocation for the offloadable methods with different resource requirements and exploits parallelism for methods execution to improve execution speed and save energy. Experimental results show that ThinkAir can achieve energy savings for an application (e.g., N-queens puzzle) of two orders of magnitude compared with local execution.

Qian et al. [2014] presented a computation offloading system, Jade, to perform dynamic fine-grained code offloading among multiple mobile devices owned by a user to minimize their energy consumption. Jade consists of three key components, as shown in Figure 14. The profiler is in charge of continuously monitoring and collecting the characteristic information of the device and the wireless network, such as CPU workload, network bandwidth, and battery level. Subsequently, based on the collected monitoring results, the optimizer makes the offloading decision to determine where to offload by solving an optimization problem. Finally, the communication manager looks for the available mobile devices to execute offloadable code and coordinates among these devices. Further, for a different amount of data transmission, Jade selects WiFi for big data transfer and Bluetooth for small data transfer. Experimental results show that Jade can achieve 74% and 86% energy savings for FaceDetection and TextSearch, respectively.

We have elaborated on some classic computation offloading frameworks; there are also some dynamic offloading methods that describe computation offloading mechanisms from other perspectives. Yang et al. [2013b] observed that CloneCloud [Chun et al. 2011] usually transfers all reachable objects from the cloud to devices, which increases the transmission time. They leveraged the compiler code analysis techniques to evaluate the possibility to compress the amount of transferred data by transferring only the essential objects. Kwon et al. [2013] presented an adaptive offloading strategy to offload the suspected energy consumption hotspots in an application specified by the programmer under fluctuating network conditions. Zhang et al. [2017a] developed a generic offloading model and a scheduling algorithm for video applications to adaptively offload tasks to the cloud in fine granularity under dynamic wireless connections. Barbarossa et al. [2013] also studied task migration mechanism under a dynamic wireless environment. They designed a cross-layer optimization framework that jointly takes application requirements, scheduling, and radio resource allocation into account to optimally allocate radio resources for the offloadable data communication. In addition to data communication, Liu et al. [2010] focused more on the privacy risks incurred by offloading. They used steganography techniques to hide offloadable data to prevent cloud servers from recognizing the real information to protect data privacy.

4.4 Summary

In this section, we summarize a variety of energy-efficient offloading strategies for mobile devices. Static or dynamic offloading mechanisms can reduce application execution time and save device power by migrating computational tasks to clouds. We divide these offloading strategies into two categories. The first category refers to the methods that offload tasks to the clouds in a fine-grained and energy-aware manner. These kinds of methods require programmers to modify application programs to achieve application partition and make offloading decisions. The second category refers to the coarse-grained computation offloading schemes. In this scenario, the entire application/process/program is migrated to the clouds. In this way, the programmers could fully leverage the efficient dynamic offloading strategies without having to modify the application source code. The energy consumption during offloading usually involves the power consumed during application execution, remote cloud communication, and other operations (e.g., data privacy and security protection for offloadable tasks). Thus, compared with local execution, offloading may not always be energy-efficient considering that the total energy for executing these operations could exceed the local execution energy. That is, offloading is feasible only when the energy consumed by local devices is more than the total energy consumed incurred by data transmission and other energy-consuming operations during offloading. When the amount of computation is large, complete offloading strategies should be adopted. Otherwise, partial offloading strategies should be adopted to avoid unnecessary data communication/transmission energy waste. Further, in order to make use of the abundant cloud resources, a better balance of energy costs between data communi-

Table 6. A Comparison Summary of Computation Offloading Strategies

Method	D_e	T_e	C_e	C_p	Off_loc	Off_parti
CloneCloud	✓	★	×	×	remote_cloud	offline
ThinkAir	✓	★	×	×	remote_cloud	online
MAUI	✓	★	×	×	remote_cloud	runtime
MACS	✓	★	×	×	remote_cloud	adaptive
Cuckoo	✓	★	×	×	remote_cloud	runtime
Phone2Cloud	✓	★	×	×	remote_cloud	offline
Jade	✓	✓	×	×	mobile_device	runtime
MCCTS	✓	✓	×	×	remote_cloud	offline
EETS	✓	✓	×	×	remote_cloud	offline
QAPTS	✓	✓	×	×	proxy	offline
FDEO	✓	✓	×	×	remote_cloud	offline
SECH	✓	✓	×	×	remote_cloud	runtime
EA-MRDA	✓	✓	×	×	remote_cloud	adaptive

Method	Deci.	Gran.	Ene.	Perf.	Benchmark	Reference
CloneCloud	dyn	thread	+20x	+20x	local_execution	[Chun et al. 2011]
ThinkAir	dyn	method	+100x	+100x	local_execution	[Kosta et al. 2012]
MAUI	dyn	method	+10x	+1x	local_execution	[Cuervo et al. 2010]
MACS	dyn	service	+90%	+20x	local_execution	[Kovachev et al. 2012]
Cuckoo	dyn	method	+40x	+60x	local_execution	[Kemp et al. 2010]
Phone2Cloud	sta	app	+86%	+91%	local_execution	[Xia et al. 2014]
Jade	dyn	class	+74%	+40%	local_execution	[Qian and Andresen 2014]
MCCTS	dyn	task	+20%	+95%	exhaustive_search	[Lin et al. 2014]
EETS	dyn	task	+82%	+26%	local_execution	[Liu et al. 2016]
QAPTS	dyn	task	+35%	÷	local_execution	[Balakrishnan and Tham 2013]
FDEO	dyn	thread	+94%	+85%	local_execution	[Yang et al. 2013b]
SECH	dyn	program	+80%	÷	local_execution	[Kwon and Tilevich 2013]
EA-MRDA	dyn	task	+15%	÷	local_execution	[Zhang et al. 2017a]

D_e: Device energy, T_e: Transmission energy, C_e: Cloud server energy, C_p: Cloud price, Off_loc: Offloading location, Off_parti: Offloading partition, Deci.: Decision, Gran.: Granularity, Ene.: Energy saving, Perf.: Performance improvement. ✓ indicates that the factor is considered in the literature. ★ implies that the literature mentions the factor but does not conduct an in-depth study of it. × indicates that the literature does not consider the factor. ÷ implies that the literature does not give a clear explanation of the factor.

cation and local execution is required. Although numerous studies have achieved tradeoff of energy between data transmission and local execution, most of them concentrate more on performance improvement (e.g., response time). The communication/transmission energy consumption between mobile devices and remote clouds plays a crucial role during offloading, thus cannot be ignored when offloading. In the next section, various wireless communication/transmission energy optimization mechanisms are introduced for saving devices' energy. Table 6 gives a summary of performance comparisons of offloading strategies.

5 REDUCE COMMUNICATION ENERGY OF DEVICES

Wireless transmission energy consumption incurred by offloading cannot be ignored under poor wireless signal. Wireless communication technology is one of the most important information transmission mediums among devices. It refers to a variety of wireless communication devices (e.g., smartphones, tabs, and laptops) and technologies ranging cellular, WiFi, Bluetooth, NFC,

Table 7. A Classification of Transmission Energy Consumption Optimizations

Classification	References
Wireless interface power model	[Pluntke et al. 2011] [Hoque et al. 2014] [Ding et al. 2013b] [Yao et al. 2013]
Communication protocols	[Yoo and Park 2011] [Pluntke et al. 2011] [Nirjon et al. 2012] [Hoque et al. 2014]
Wireless network interfaces' characteristics	[Qian and Andresen 2014] [Yoo and Park 2011] [Pluntke et al. 2011] [Ding et al. 2013a] [Nirjon et al. 2012] [Donohoo et al. 2012] [Deng and Balakrishnan 2012] [Dogar et al. 2010] [Hoque et al. 2014] [Bui et al. 2013] [Ding et al. 2013b] [Kosta et al. 2013]
Intermittent/unstable wireless network connectivity	[Guo and Liu 2018] [Chun et al. 2011] [Kosta et al. 2012] [Kwon and Tilevich 2013] [Zhang et al. 2017a] [Barbarossa et al. 2013] [Bui et al. 2013] [Li et al. 2013a] [Zhang et al. 2013] [Zhang et al. 2015a] [Tong and Gao 2016] [Yao et al. 2013] [Zhang et al. 2015b] [Kwon and Tilevic 2012] [Terefe et al. 2016] [Goudarzi et al. 2017] [Shi et al. 2012] [Magurawalage et al. 2014] [Gai et al. 2016] [Zhou et al. 2015] [Shi et al. 2014]
Device sleep (doze off)	[Dogar et al. 2010]
Data compression	[Shu et al. 2013] [Yao et al. 2013]

and the like. Modern wireless devices usually include cellular networks (e.g., 2G, 2.5G, 3G, and 4G), Bluetooth, and WiFi. Bluetooth is a peer-to-peer wireless technology for data communication between mobile devices over short distances. Due to the short operating range, Bluetooth is fairly low power. WiFi is known as a Local Area Network (LAN) technology based on IEEE 802.11 standards due to its moderate coverage area. Using WiFi, mobile devices connect to the Internet via a Wireless LAN and a wireless access point (AP). In case of a mobile device that needs access to the clouds—if it is not located near a WiFi AP—then it may need a cellular radio (e.g., 2G, 2.5G, 3G, and 4G) for long distance communication. Overall, Bluetooth consumes much less power than WiFi, and a lot less power than cellular technologies, but still significantly more than technologies such as Bluetooth Low Energy (BLE) and Zigbee, which are short-range networking technologies.

In telecommunication, a communication protocol (e.g., Transmission Control Protocol (TCP)) refers to a system of rules allowing two or more devices to transmit information [CP 2019]. Traditional TCP protocol may be energy-inefficient for transfering data. It is necessary to design or utilize more energy-efficient transmission protocols to achieve energy savings transmission between mobile devices and remote clouds (Section 5.1). We have known that different wireless communication interfaces (e.g., Bluetooth, WiFi, 2G, 3G, and 4G) usually consume different amounts of energy. In other words, these interfaces may achieve energy optimal transmission under different conditions. This characteristic enables a new insight to save device energy by making these interfaces work in tandem (Section 5.2). Taking into account the fact that the wireless network connectivity is usually unstable and intermittent, it is feasible to transmit most of the offloaded parts under good connection conditions to avoid significant transmission energy consumption incurred under poor connection conditions (Section 5.3). Table 7 summarizes strategies for reducing wireless transmission energy consumption.

5.1 Utilize Energy-Efficient Protocols

Yoo et al. [2011] noticed that Wireless Local Area Network (WLAN) interfaces consume greater energy than Bluetooth interfaces in mobile devices. Thus, they explored energy-saving

possibilities by utilizing WLAN and Bluetooth collaboratively. A distributed group mechanism called Cooperative Networking protocol (CONET) is designed to reconfigure the groups dynamically based on each device's bandwidth, energy, and application requirements. A group is a Bluetooth Personal Area Network (PAN) that contains one head node and a couple of common nodes. The head node is regarded as a gateway between PAN and WLAN, and is responsible for forwarding packets from clusters to WLAN. In this way, the common nodes can turn off the WLAN interfaces and access the WLAN infrastructure through low-power Bluetooth. Experimental results show that compared with WLAN-only communication, their proposed CONET can achieve 25% device energy reduction.

Pluntke et al. [2011] found that Multipath TCP (MPTCP) is an efficient mechanism to switch flow among multiple network paths without breakage. This characteristic motivates them to explore the energy-saving ideas by selecting the most energy-efficient wireless network interface available on mobile devices. They designed a novel multipath scheduling algorithm to achieve energy savings by adaptively controlling wireless interface switching according to current connectivity conditions. Specifically, based on the application model and the wireless interfaces' energy models, the scheduling problem is formulated as a Markov decision process (MDP) to obtain the optimal schedule schemes automatically. Experimental results show that compared with the oracle scheduler, their proposed multipath scheduling algorithm can help a device achieve up to 23% and 9% energy savings for 3G and WiFi, respectively.

5.2 Leverage Characteristics of Wireless Interfaces

Ding et al. [2013a] noted that the number of WiFi APs is limited for mobile devices to affect the computation offloading opportunities. They designed a cooperating WiFi-based offloading architecture to achieve energy savings for mobile devices. This architecture leverages the combination of cellular, WiFi, and mobile users to make them work collaboratively to save energy. An energy-aware algorithm is presented to make offloading decisions by selecting the most energy-efficient WiFi AP for computation offloading. For example, mobile devices may start offloading data over the 3G interface but try to find WiFi APs at the same time to prefetch and transfer data with lower energy overheads. Experimental results show that their method can reduce 80% device energy consumption when offloading mobile traffic to WiFi networks.

Similar to the ideas presented in Pluntke et al [2011], Ding et al. [2013a], and Nirjon et al. [2012] investigated energy saving opportunities by dynamically switching between wireless interfaces based on the TCP characteristics. Donohoo et al. [2012] also developed a strategy to adaptively manage wireless interfaces according to the spatiotemporal and device context. However, different from Nirjon et al. [2012], they leveraged machine learning algorithms to learn the users' usage patterns of mobile devices to predict the configurations of wireless interfaces. Based on the prediction results, the wireless interface states are managed dynamically to achieve the optimal energy configurations. Deng et al. [2012] also adopted machine learning algorithms to manage wireless interfaces. However, the contents they learned and predicted are network traffic activities. Based on the predicted network traffic activities, their proposed traffic-aware technique can control the state transitions of a 3G/LTE radio based on their energy consumption. Dogar et al. [2010] considered reducing transmission energy from another aspect, that is, allowing mobile devices to sleep when transferring data. They designed a system, Catnap, to achieve this goal by leveraging the bandwidth discrepancy between the wireless and the wired links to look for sleep opportunities. Experimental results show that Catnap can prolong the battery lifetime by up to 5x by allowing mobile devices to sleep.

Hoque et al. [2014] reduced the wireless communication energy of mobile devices by forming the multimedia traffic into periodic bursts when accessing multimedia services. To this end, they

explored the relationship between burst size and the power consumed by wireless network interfaces in mobile devices. They found that the power consumption of these wireless interfaces shows different characteristics when the burst size of traffic and buffer size on mobile devices do not match properly. According to this observation, an energy model for wireless interfaces is designed to analyze the energy consumption of bursty traffic. Finally, a power-frugal multimedia delivery system, EStreamer, is proposed to decide an energy-optimal burst size according to the wireless interfaces energy model. Experimental results show that their method can extend 3x battery lifetime by forming the multimedia traffic into periodic bursts.

5.3 Adapt to Intermittent Wireless Connectivity

Bui et al. [2013] noted that mobile devices are energy-consuming when aggregating bandwidth over multiple wireless interfaces. They designed an energy-efficient bandwidth aggregation middleware, GreenBag, to support data-streaming services over LTE and WiFi links. GreenBag uses a link management mechanism to dynamically deliver data streaming in an energy-efficient way under varying wireless networks. More specifically, it first uses a segmentation method to split data. Subsequently, it performs load balancing between LTE and WiFi links. Then, based on current wireless connectivity conditions, it predicts the future wireless connectivity conditions and determines the corresponding load ratio between LTE and WiFi links for each data segment. Finally, GreenBag conducts a cut-off policy to opportunistically stop the use of a redundant link to achieve energy savings on mobile devices. Experiment results show that GreenBag can reduce about 25% energy consumption as compared with non-energy-aware methods.

Shu et al. [2013] developed a strategy, eTime, to transmit data in an energy-efficient way under intermittent wireless connections. Encouraged by prefetching-friendly or delay-tolerant applications, eTime first manages applications' data by configuring resources in the cloud. Then, based on current wireless network conditions, it makes transmission decisions dynamically by using Lyapunov optimization techniques to balance energy consumption and application execution delay. It looks for the good opportunities to prefetch the commonly used data under good connection conditions and delay the transmission of delay-tolerant data under bad connection conditions. Zhang et al. [2013] also presented an optimization framework to judiciously execute applications locally or remotely to save device energy under stochastic wireless channel conditions. On the device side, the energy consumed by computation is optimized through adapting the CPU frequency according to the workload dynamically. On the cloud side, the transmission energy consumption is optimized by adapting the data transmission rate to the stochastic wireless channels. These two scheduling problems are first formulated as constrained optimization problems. Subsequently, they obtained closed-form solutions for the optimal scheduling strategies. Experimental results show that their schemes can achieve 35% and 13x energy savings as compared with a random strategy and local execution, respectively.

Tong et al. [2016] proposed an application-aware transmission scheduling scheme by adaptively scheduling workloads between mobile devices and remote clouds based on the characteristics of the causality and the runtime dynamics of applications. First, considering the causality of an application, an efficient offline scheduling algorithm is designed to dispatch the workloads based on a predetermined transmission order. Further, taking into account the dynamics of the application running as well as predicting the future execution paths of the application, a stochastic scheme is presented to extend the offline scheduling mechanism to online scheduling. Experimental results show that when the application delay constraint is tight, their method can save 40% energy as compared with Bundle transmission [Zhao et al. 2013].

Ding et al. [2013b] investigated the effect of wireless connection conditions on the energy consumed by mobile devices. Usually, poor signal strength may cause data retransmission and increase

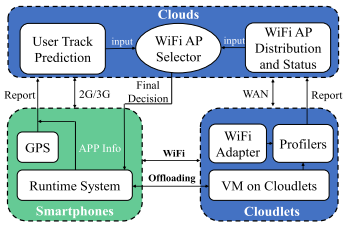


Fig. 15. The architecture of ENDA [Li et al. 2013a].

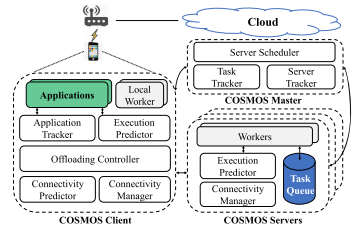


Fig. 16. The architecture of COSMOS [Shi et al. 2014].

transmission time, incurring more transmission energy consumption. Thus, they measured the additional energy consumption for data transmission due to the weak wireless signal strength. Based on the measurement results and the signal strength factor, an online power model is designed for WiFi and 3G to analyze the power behaviors of the applications or services at runtime. This power model can explore the dynamics of the wireless signal strength to save interface energy by delaying data transmission until a good wireless signal strength is encountered for delay-insensitive applications or services. Experimental results show that their method can save up to 23.7% and 21.5% transmission energy under WiFi and 3G, respectively.

Satyanarayanan et al. [2009] investigated the next-generation alternative technologies to handle the defects existing in resource-constrained mobile devices and distant-delayed cloud computing when running immersive applications. They introduced a new computing architecture called cloudlet, a small cloud data center with rich resources at the Internet edge to provide services for nearby mobile devices. In this architecture, mobile users quickly instantiate custom service software on nearby cloudlets using virtual machine (VM) technology and then use the service over a wireless LAN. Based on cloudlets, Li et al. [2013a] presented a novel architecture, ENDA, to make computation offloading decisions under intermittent wireless network conditions. As shown in Figure 15, ENDA considers three computing resources including mobile devices, cloudlets, and clouds. In this context, these three computing resources are interacted with each other to collaboratively perform offloading under dynamic network conditions. In particular, most operations of decision-making are placed on the remote clouds and cloudlets. On the cloud tier, a greedy searching mechanism is proposed to forecast users' behavior activities based on the historical data stored in servers. Then, a cloud-enabled WiFi AP selection mechanism is presented to find the most energy-efficient access point, i.e., cloudlets, to offload for mobile devices. Experimental results show that ENDA can achieve optimal offloading decisions for almost all data sizes while existing random methods can only produce the most energy-efficient decision with a 1/3 probability.

Shi et al. [2014] designed COSMOS, a system that provides dynamic computation offloading mechanisms for mobile devices. As shown in Figure 16, COSMOS consists of three main components. The COSMOS Master is responsible for managing cloud resources and exchanging information with mobile devices. The COSMOS Server is in charge of executing the offloaded tasks. The COSMOS Client is responsible for monitoring the application executions and the wireless network conditions and making offloading decisions. In order to achieve the computation offloading benefits, they presented a risk-control offloading mechanism. The risk-control offloading mechanism can properly assess the "returns" and the "risks" for offloaded tasks when making offloading decisions. Specifically, when a task is initiated, the offloading controller evaluates its return and risk before performing data migrating. If the risk-adjusted return is larger than a threshold, the offloading controller will send the task to the cloud. Otherwise, the task is executed locally. Moreover, the risk-control offloading mechanism also monitors the network connectivity status, which has an

important effect on the assessments of return and risk when offloading tasks. If network connectivity changes, the offloading mechanism will re-calculate the risk-adjusted return and dynamically vary its offloading decision. Experimental results show that compared with CloneCloud [Chun et al. 2011] assuming stable network connectivity, COSMOS can save about 10x cost for offloading.

5.4 Summary

The issues of frequent network disconnection, low bandwidth access, unstable network conditions, and fragile wireless communication security are often happening in wireless communication. In this section, wireless transmission energy consumption for mobile devices is optimized from low-power communication protocols, wireless network interfaces' power characteristics, and adaptive transmission under dynamic network conditions. In particular, data compression [Yao et al. 2013] is also an efficient method for transmission energy optimization. Usually, when offloading to remote clouds, long-distance and low-power communication techniques are needed. In a low-power wide-area network (LPWAN), low-bit-rate long-distance data communication can be achieved among mobile devices. This characteristic makes LPWAN more energy-efficient than a WAN when performing offloading. Other available energy-efficient wireless technologies such as Long Range (LoRa) [A. Augustin and Townsley 2016], Narrowband IoT (NB-IOT) [R. Ratasuk and Ghosh 2016; R. Sinha and Hwang 2017], and LTE Machine type communication (LTE-M) [U. Raza and Sooriyabandara 2017] also offer good solutions to long-distance offloading problems. However, none of these wireless technologies are widely supported global standards, thus making their implementation challenging and even impossible. In general, transmission energy can be saved from three points. First, select the optimal wireless interface based on where the data or method is offloaded. If possible, try to make these wireless interfaces work in a collaborative manner for offloading. Second, compress the offloaded parts of applications as much as possible to reduce the amount of data transferred under the premise of not affecting applications' normal operations. Third, adaptively adjust offloading strategies when offloading data to the cloud based on wireless network conditions to avoid energy waste in poor wireless connectivity scenarios. Data security and privacy protection is also an important concern when transmitting data over the network. Usually, these issues can be addressed from a mobile device side, cloud data centers side, and data transmission process. A common way to prevent offloaded data from being attacked is to encrypt offloaded data using encryption techniques. However, encryption operations will incur additional energy consumption for devices and affect offloading decisions [Kumar and Lu 2010]. MEC is emerging as an effective technique for optimizing wireless communication energy consumption between devices and remote clouds by providing offloading services closer to users. For easy classification and presentation, we put the contents of MEC/FC in Section 6. In the next section, we introduce various cloud execution mechanisms. Table 8 gives a summary of the performance comparisons of transmission energy optimizations.

6 EXECUTE OFFLOADED COMPUTATION IN CLOUDS

Numerous cloud-assisted MC platforms have been investigated to achieve computation offloading. These cloud-assisted mobile platforms can be classified into three categories, namely, infrastructure-based remote clouds (e.g., CloneCloud), infrastructure-based edge/fog clouds (e.g., Cloudlets, mobile edge servers, and fog servers), and ad-hoc virtual clouds (e.g., mobile device cloud). No matter which cloud-assisted mobile computing platform, they are all equipped with powerful servers that possess richer computing and storage resources, as well as powerful processing capacity compared to mobile devices. Usually, a entire offloading process can be described simply as follows. Supposing that a mobile device is running an application including three methods (i.e., method A, B, and C). In particular, the Method B is a computation-intensive method that

Table 8. A Comparison Summary of Transmission Energy Optimizations

Method	BT	3G	4G	WiFi	Com_prot	WireInter_P_m	Comments
CONET	✓	×	×	✓	TCP	×	distributed clustering
MPTCP	×	✓	×	✓	TCP	✓	multipath TCP
MADNet	×	✓	×	✓	TCP	×	mobile traffic offloading
Catnap	×	×	×	✓	TCP	×	device sleep
EStreamer	×	✓	✓	✓	TCP	✓	rate-adaptive streaming
GreenBag	×	✓	✓	✓	TCP	✓	bandwidth aggregation
eTime	×	✓	×	✓	÷	✓	prefetch data
EOOF	×	×	×	×	÷	×	stochastic wireless channel
AATS	×	✓	✓	×	÷	✓	application-aware transfer
WSS	×	✓	✓	✓	TCP	✓	model signal strength
ENDA	×	✓	×	✓	÷	×	WiFi AP selection
MCS	×	×	×	×	÷	×	mobile cloudlet systems
COSMOS	×	✓	×	✓	÷	×	reduce offloading cost
EETS	×	×	×	✓	÷	×	data compression

Method	Off.	Off_parti	Deci.	Gran.	Ene.	Benchmark	Reference
CONET	×	×	×	×	+25%	wlan_only	[Yoo and Park 2011]
MPTCP	×	×	×	×	+23%	3G_only	[Pluntke et al. 2011]
MADNet	✓	adaptive	dyn	traffic	+80%	3G_only	[Ding et al. 2013a]
Catnap	×	×	×	×	+2x∼+5x	non_sleep	[Dogar et al. 2010]
EStreamer	×	×	×	×	+1.5x∼+3x	no EStr.	[Hoque et al. 2014]
GreenBag	×	×	×	×	+14%∼+25%	non_energy-aware	[Bui et al. 2013]
eTime	×	×	×	×	+20%∼+35%	random strategy	[Shu et al. 2013]
EOOF	✓	×	dyn	app	+13x	mobile execution	[Zhang et al. 2013]
AATS	✓	adaptive	dyn	thread	+40%	bundle transmission	[Tong and Gao 2016]
WSS	×	×	×	×	+23.7%	poor signal strength	[Ding et al. 2013b]
ENDA	✓	×	dyn	app	Pro: 1/3	random WiFi APs	[Li et al. 2013a]
MCS	✓	adaptive	dyn	code	+4% (cost)	local execution	[Zhang et al. 2015a]
COSMOS	✓	adaptive	dyn	task	+10x (cost)	CloneCloud	[Shi et al. 2014]
EETS	✓	adaptive	dyn	task	+99% (accuracy)	non-offloading	[Yao et al. 2013]

BT: Bluetooth, Com_prot: Communication protocol, WireInter_P_m: Wireless interface power model, Off.: Offloading, Off_parti: Offloading partition, Deci.: Decision, Gran.: Granularity, Ene.: Energy saving. ✓ indicates that the factor is considered in the literature. × indicates that the factor is not considered in the literature. ÷ implies that the literature does not give a clear explanation of the factor.

needs to be migrated from mobile devices to cloud servers for execution. Correspondingly, methods A and C are executed on local mobile devices. Based on current network conditions, the mobile device will make dynamic offloading decisions for method B to reduce transmission energy consumption. And the cloud schedules its powerful virtual computation resources (e.g., servers) for executing the offloaded components (i.e., method B). Once the cloud completes the task execution, it returns and merges the execution result of method B with that of the methods A and C executing on the local mobile device.

In this section, we take a far-to-near sequence to describe the execution mechanisms of mobile cloud applications in these cloud-assisted platforms. The infrastructure-based remote clouds are empowered by device clones and execution engines in cloud servers. Remote clouds replace the local mobile devices to execute application components for battery lifetime extension of these mobile devices (Section 6.1). The infrastructure-based edge/fog clouds execute the offloaded

Table 9. A Classification of Execution Mechanisms in Clouds

Classification	References
Remote servers	[Guo and Liu 2018] [Shu et al. 2013] [Cuervo et al. 2010] [Kovachev et al. 2012] [Lin et al. 2014] [Liu et al. 2016] [Balakrishnan and Tham 2013] [Kemp et al. 2010] [Xia et al. 2014] [Liu et al. 2010] [Kwon and Tilevich 2013] [Zhang et al. 2017a] [Barbarossa et al. 2013] [Ding et al. 2013a] [Zhang et al. 2013] [Tong and Gao 2016] [Yao et al. 2013] [Kwon and Tilevic 2012] [Wang et al. 2013] [Terefe et al. 2016] [Goudarzi et al. 2017] [Ravi and Peddoju 2015] [Zhou et al. 2015]
Remote cloud clone	[Chun et al. 2011] [Kosta et al. 2012] [Barbera et al. 2013] [Yang et al. 2013b] [Zhang et al. 2015b] [Kosta et al. 2013]
Edge servers	[Guo and Liu 2018] [Zhang et al. 2015a] [Beck et al. 2015] [Fan et al. 2017] [Zhang et al. 2017b] [Zhang et al. 2018] [Chen et al. 2018] [Habak et al. 2015] [Wu et al. 2016] [Gai et al. 2016] [Ravi and Peddoju 2015] [Zhou et al. 2015] [Shi et al. 2014]
Mobile device cloud	[Tian et al. 2018] [Qian and Andresen 2014] [Shi et al. 2012] [Mtibaa et al. 2014] [Mtibaa et al. 2013b] [Mtibaa et al. 2013a] [Habak et al. 2015] [Wu et al. 2016]
Collaborative execution	[Guo and Liu 2018] [Li et al. 2013a] [Zhang et al. 2015b] [Kwon and Tilevic 2012] [Wang et al. 2013] [Terefe et al. 2016] [Habak et al. 2015] [Wu et al. 2016] [Magurawalage et al. 2014] [Gai et al. 2016] [Ravi and Peddoju 2015] [Zhou et al. 2015] [Rahimi et al. 2012]

computational tasks of an application by employing mobile edge/fog servers to co-locate with the nearby base stations to overcome long-distance data transmission incurred by remote clouds (Section 6.2). The ad-hoc virtual clouds consists of a group of nearby mobile devices that work in tandem to accomplish computation offloading when the remote and edge/fog servers are unavailable (Section 6.3). In general, each of the above cloud-assisted mobile platform can independently complete the computation offloading process and relieve the burden on energy-limited mobile devices. However, as stated in Zhang et al. [2015a], the issues of intermittent wireless networks may exist considering users' mobility, complex network, and diverse wireless connectivity mechanisms, thus may fail the whole offloading process. Thus, it is necessary for multiple cloud-assisted mobile platforms to complete the offloading process collaboratively to achieve energy savings for mobile devices (Section 6.4). Table 9 summarizes the application/task execution mechanisms in diverse mobile cloud-assisted platforms.

6.1 Execute Computation in Remote Clouds

Zhang et al. [2015b] performed offloading between mobile devices and remote clouds to save device energy. In this context, the cloud possesses a device clone for each mobile device, and an application is constructed as sequential tasks that can be processed locally or remotely. In this study, an application has n tasks; the data sizes of the input and the output for each task ω_i are represented by α_i and $\beta_i (i = 1, 2, \ldots, n)$, respectively. The goal is to judiciously offload the tasks between mobile devices and remote clouds under stochastic wireless channels. They considered three channel models that have a different channel state. According to the context of each task and the channel state, a task is assigned to be executed locally or sent to the remote clouds for execution. In particular, the task execution process is formulated as a shortest path problem with constraints, which is subsequently solved by either the enumeration algorithm or the Lyapunov optimization

to obtain the optimal or sub-optimal solutions, respectively. Simulation results show that compared with local and remote execution, their method can save 5x and 20% energy, respectively.

Kwon et al. [2012] developed an energy-efficient strategy to perform offloading without having to partition an application to prevent network power outages. Similar to CloneCloud [Chun et al. 2011], the proposed strategy offloads the power-intensive parts to the cloud for execution. Similar to MAUI [Cuervo et al. 2010], the proposed strategy uses checkpointing to synchronize the states between mobile devices and remote clouds. A clear difference of the proposed strategy from the above two methods is that it can execute the power-intensive parts of an application in the cloud without needing to conduct application partition. Specifically, it just needs to replicate the application execution states that switch between the mobile devices and the remote clouds to save device energy and deal with the case of network outages. When the wireless network is connected, the power-intensive portions of the application will be offloaded by delivering only the needed states to the remote clouds. Otherwise, the execution in the remote clouds will be redirected back to the mobile devices. Experimental results show their approach can help applications consume up to 60% fewer Joules than their original versions.

Kosta et al. [2013] investigated computation offloading between mobile devices and the clouds, and device-to-device communication offloading in the clouds. They proposed a distributed computing platform, Clone2Clone, as shown in Figure 17 to save device energy. In this platform, each device has its clone in the cloud, and these device clones are connected in a peer-to-peer way to exploit networking services. Specifically, CloneDS is responsible for mapping not only a mobile user to a clone but also a clone to an IP. Each newly created clone first needs to register and lookup from the CloneDS. Then, it starts Peer-to-Peer connections with other clones and its IP can be obtained through a CloneDS lookup by mobile users. Finally, mobile users connect to its clone through its public IP, and install any application in their cloned devices. In general, Clone2Clone puts computation and communication tasks in the cloud, which greatly reduces the computation and communication overheads on mobile devices. Experimental results show that Clone2Clone can achieve 30% energy savings for document editing as compared with SPORC [Feldman et al. 2010].

Wang et al. [2013] considered such a scenario that multiple homogeneous servers in a cloud are dedicated to processing offloaded tasks from multiple mobile devices simultaneously. They proposed a two-stage game formulation method to solve the offloading issues. In the first stage, mobile devices decide which parts of the service requests should be sent to the remote clouds to minimize their power consumption. In the second stage, according to the arrival rate of the service requests, these requests are assigned to a server by the cloud controller, and parts of the computing resources of the server are allocated to process these service requests. Finally, the convex optimization technique is leveraged to solve this two stage game formulation to obtain the optimal offloading strategy between mobile devices and the remote clouds. Experimental results show that their method can save 21.8% energy as compared with local execution.

Terefe et al. [2016] offloaded computation to multiple heterogeneous servers with different capabilities. They developed an offloading method that differentiates data-intensive and computation-intensive components of applications, and offloads them to appropriate servers. Specifically, they first proposed two energy models for applications and wireless network channels, respectively. Then, the multi-server partitioning problem is formulated by using a MDP framework. Finally, an energy-efficient multi-server offloading strategy algorithm is designed based on a value iteration method. This algorithm obtains an efficient solution to the multi-server partitioning problem. Likewise, Goudarzi et al. [2017] studied heterogeneous multi-site application offloading between mobile devices and remote clouds to balance energy consumption and execution time. An application is represented by a relation graph that contains vertices and edges.

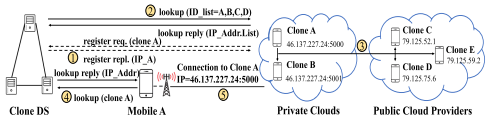


Fig. 17. The architecture of Clone2Clone [Kosta et al. 2013].

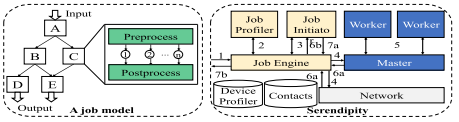


Fig. 18. The architecture of Serendipity [Shi et al. 2012].

Vertices refer to the constituent parts of the application while edges refer to the invocations among these constituent parts. They first presented a weighted cost model to describe the local and remote execution overheads (vertices) and invocations costs (edges) by online or offline profiling. Then, a multisite offloading scheme based on application size is designed to solve the offloading problem. For small-scale applications, the branch-and-bound method is leveraged to obtain the optimal solution. For large-scale applications, they used particle swarm optimization to achieve a near-optimal solution. Experimental results show that their method can save 38% energy as compared with Multi-factor Multi-site Risk-based Offloading (MMRO) [Wu and Huang 2014].

6.2 Execute Computation in Edge Clouds

Instead of offloading to the remote clouds, Beck et al. [2015] saved transmission energy consumption by offloading the applications or tasks to the edge servers that are deployed on nearby base stations. In this context, they investigated energy-efficient video encoding offloading during video calls. They presented an MEC-based video telephony system, called Mobile Edge-Voice over Long Term Evolution (ME-VoLTE), to offload the encoding efforts to the nearby edge servers. In ME-VoLTE, mobile devices first select low-compression optimization to reduce the encoding efforts. Subsequently, the video data is transferred to the edge servers by using encoding techniques, which are less computation-intensive and more energy-efficient. Finally, the edge servers leverage a more powerful and higher compression ratio codec to transcode the video data on behalf of the mobile devices before forwarding the data to the remote participants. Experimental results show that ME-VoLTE can achieve 13% power savings for mobile devices as compared with local execution.

Fan et al. [2017] noted that a single mobile edge server cannot process the increasing tasks offloaded from the mobile devices in time. They focused on the joint offloading problem among multiple mobile edge servers to manage device energy. To achieve this, a novel cooperative offloading strategy is proposed to augment the computation offloading ability of the original edge server by further offloading the extra tasks to other edge servers. Specifically, the original edge server first sends the offloading request (e.g., necessary task information) to the target edge server. Then, the target edge server processes the corresponding tasks, and eventually returns the processing results to the original edge server. Simulations results show that compared to a non-cooperation scheme, their proposed offloading strategy can achieve 45% improvement of total benefits (i.e., execution time and energy consumption). Zhang et al. [2017b] also considered computation offloading in the context of multi-cell MEC. They proposed a dynamic offloading strategy to simultaneously optimize the allocation of transmission channels and edge computing resources to balance energy consumption and execution latency. For a single server, they formulated and solved the offloading problem by computing the local and edge execution overheads separately, then compared them to make the optimal offloading decision. For multiple servers, they formulated the offloading problem as a mixed integer nonlinear problem, and proposed an iterative search algorithm to solve this non-convex problem and obtain a suboptimal solution. Experimental results show that compared with local execution and edge server execution, their method can achieve 63% and 57% cost savings, respectively.

Zhang et al. [2018] also balanced the energy consumption and the task execution latency. However, their proposed online dynamic offloading scheme novelly utilizes the energy harvesting devices to collect energy from the circumambient environment to power mobile devices. Their scheme can optimize the energy consumption and execution delay while maintaining stable battery level. The scheme uses the Lyapunov optimization method to make offloading decisions (e.g., local execution, edge execution, or drop) considering the information related to tasks, network condition, harvestable energy, and device energy. Likewise, Chen et al. [2018] also adopted energy harvesting devices. However, these energy harvesting components are deployed on edge servers rather than mobile devices to collect ambient renewable energy. In such a mobile edge environment, multiple mobile devices attempt to perform computation offloading to the edge servers simultaneously. They formulated this multi-device offloading problem and also utilized the Lyapunov optimization method to solve and obtain the optimal energy harvesting strategy, task scheduling, and resource allocation schemes. Experimental results show that their methods can save 14% device energy and achieve up to 31.9% higher system utility as compared to CPU-only and a random scheduling scheme, respectively.

6.3 Execute Computation in Mobile Device Clouds

Shi et al. [2012] considered such a scenario that mobile devices are connected with each other to perform offloading. They designed a system, Serendipity, as shown in Figure 18, to offload computation to the available mobile devices to save energy. Specifically, after receiving a job submitted by a user, the job engine uses the job profiler to construct a job profile to evaluate tasks' executive time and energy consumption on every mobile node. Meanwhile, the job initiator constructs a job model that contains multiple PNP-blocks, which is the basic job component for offloading. Then, the job engine offloads the tasks to either the local master or other mobile nodes for execution based on wireless connectivity characteristics. A worker is responsible for executing a task; after its execution, the master returns the results to the job initiator, which will then trigger a new PNP-block. After processing all the tasks, the job initiator returns the final results and the offloading process finishes. Experimental results show that Serendipity can achieve up to 6.6x speedup and complete more jobs in the same amount of time compared with local execution.

Mtibaa et al. [2014] introduced an offloading framework including task manager, task offloader, and task forwarder to offload computation among mobile devices. In particular, after receiving a task from an application, the manager stores the status and the characteristics of the task. The offloader decides whether and what computation to migrate based on the interaction with the privacy and security engine and the execution overhead comparison between the local and other devices. The forwarder updates the available connections to nearby devices and stores the historical contact information and social information to infer expected connections. They used an experimental approach in Mtibaa et al. [2013b] to estimate the opportunistic offloading energy consumption. Then, a data driven approach is adopted to compare multiple offloading algorithms. Finally, based on the social information shared by devices' users [Mtibaa et al. 2013a], a social-based algorithm is designed to select the most appropriate mobile node for offloading. Experimental results show that their method can achieve 4x energy savings compared with local execution.

Tian et al. [2018] leveraged reinforcement learning and DVFS techniques to optimize energy consumption among multiple mobile devices. As shown in Figure 19, for each mobile device, the interaction between the local power manager and processors constitutes a learning process. Specifically, the DVFS controller determines the dynamic Voltage/Frequency (V/F) level according to the processors' feedback state information. Further, the control strategy (V/F level) of the DVFS controller is sent to the cloud at regular intervals to deliver information to other mobile devices. The cloud is responsible for collecting the local control strategies from the whole system and com-

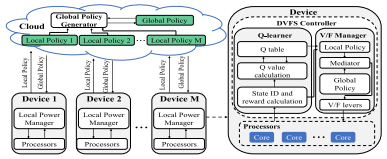


Fig. 19. Multi-device collaborative [Tian et al. 2018].

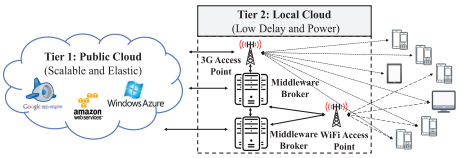


Fig. 20. The architecture of MAPCloud [Rahimi et al. 2012].

bining them to form a global control strategy, which will subsequently be shared with the mobile devices in this system. Finally, each mobile device decides its control scheme according to the local and global strategies. Experimental results show that their method can achieve 10% device energy savings as compared to learning-based approaches.

6.4 Execute Computation in Hybrid Clouds

Habak et al. [2015] presented femtocloud containing multiple mobile devices and a cloudlet to provide a self-configuring mobile device cloud. Mobile devices periodically calculate their available computation resources and send this information to the control device (cloudlet). Meanwhile, users' behavior data are collected by the profiling module to predict the device presence time in this context. For the control device (cloudlet), after receiving the arriving tasks, it predicts the presence time of each device, the execution workload of each task, and the new participated available devices. Then, based on the above information, the task scheduling module is in charge of mapping the tasks to the mobile devices. Likewise, Wu et al. [2016] also combined mobile devices and cloudlets to perform computation migration collaboratively. In this context, the cloudlet provides cloud services with lower latency at the edge while mobile devices work in tandem to achieve energy-efficient computing. In particular, the application and task models are the same as the models in Zhang et al. [2015a] and Shi et al. [2012], respectively. When the cloudlet is unavailable due to insufficient computing resources, a central scheduler will control the mobile devices to provide offloading collaboratively. The central scheduler assigns the tasks to the reliable mobile devices by using a centralized scheduling method to reduce energy consumption. Experimental results show that compared with local and single resource execution, their method can save 72% and 65% energy, respectively.

Magurawalage et al. [2014] noted that wireless network connectivity may be unavailable when performing offloading. They presented a system architecture that contains mobile devices, cloudlets, and remote clouds. This architecture dynamically determines the offloading location when conducting computation offloading. Specifically, they first proposed an offloading method to determine the offloading location (i.e., a cloud clone or a cloudlet) taking into account the task energy characteristics and wireless network conditions. Subsequently, at cloudlets, a data caching method is adopted to further enhance the system performance. Gai et al. [2016] also considered such a system architecture to reduce additional wireless communication energy consumption. However, they used dynamic programming techniques at cloudlets to dynamically predict and allocate computing resources in the remote cloud for mobile devices based on the changing operational environment. Experimental results show that their methods can save 90%, 60%, and 58% energy as compared to mobile only, clone only, and traditional cloud methods, respectively.

Ravi et al. [2015] also proposed a cooperative offloading system to select appropriate computing resources (i.e., mobile devices, cloudlets, or remote clouds) for computation offloading. The computing resource that consumes lower energy will be chosen to provide offloading services for mobile devices. This resource chosen decision is made by a multi-criteria decision-making method

taking into account many factors. Zhou et al. [2015] developed an offloading strategy among these computing resources in a dynamic network environment to optimize energy consumption. More specifically, they presented models for these computing resources to estimate their cost when performing tasks offloading. Based on the above estimation results and the dynamic device context, they designed a context-aware method to make decisions of where, when, and how to offload an application. Experimental results show that their method can achieve up to 55.6% energy savings as compared with local execution.

Rahimi et al. [2012] introduced a hybrid 2-tier computation offloading architecture, MAPCloud, which saves device energy by collaboratively leveraging the resources of the local and public clouds. As shown in Figure 20, the local clouds with the non-scalable resources are closer to the mobile users for providing better performance via WiFi, while the public clouds with high latency and power consumption provide the scalable resources for applications via 3G. In particular, they designed a middleware for MAPCloud to implement mapping between tasks and resources. The middleware preserves and maintains the resources and services available in the local and public clouds. For every arrival application in the form of a workflow of tasks, the middleware looks up the available resources or services in the local and public clouds, and utilizes admission control to check whether the tasks of this application are schedulable. If schedulable, the tasks are scheduled onto the specific nodes using a scheduling algorithm. Further, since the resource allocation optimization problem is NP-hard, a simulated annealing-based heuristic method called Cloud Resource Allocation for Mobile Applications (CRAM) is presented to derive a near optimal solution to the "task-resource" assignment problem. Simulation results show that CRAM can save 32% device power as compared with the public cloud only scheme.

Guo et al. [2018] noted that current hosted networks of MEC usually adopt network technologies with single access mode. They proposed a novel network architecture that utilizes hybrid fiber-wireless (FiWi) networks to offload in an energy-efficient way. In this context, the collaborative offloading problem is studied among mobile devices, edge servers, and cloud servers. They formulated the collaborative offloading problem as a constrained optimization problem and proposed an approximation offloading scheme to solve this problem. The scheme first computes the processing time of each task in three computing modes separately. Subsequently, based on the processing time calculation results, some tasks that cannot be processed locally are selected for edge or remote execution considering their energy consumption. Finally, for the residual tasks that can be processed in either of three computing modes, the amount of wireless channels is updated iteratively, and the corresponding energy consumption is computed separately under three computing modes. The computing mode that has the lowest energy consumption is selected to process the tasks. Further, another distributed offloading scheme based on game theory is proposed for the scenario with multiple wireless base stations. In this context, each mobile device is regarded as a game player, and carries out their current optimal offloading strategies by observing other players' offloading decisions. Experimental results show that their method can save 20% energy as compared with the distributed computation offloading scheme [Chen et al. 2016].

6.5 Summary

In this section, the application/task execution mechanisms in diverse cloud platforms have been studied in detail. Remote clouds provide sufficient computing and storage resources for executing the offloaded application components. Nevertheless, the long distance for data transmission between the mobile devices and the remote clouds incurs significant transmission energy consumption. Edge/fog clouds solve the above issues by making the computing and storage resources closer to the mobile users. However, a single edge/fog server cannot process all the arrived tasks due to its limited computing and storage resources compared with rich and powerful remote cloud

servers. Thus, collaborative execution between multiple edge/fog servers is an effective method to take away the pressure on the single edge/fog server. Intermittent and unstable wireless connectivity may cause edge or remote servers to be unavailable to mobile devices. At this time, adjacent mobile devices can work in tandem to execute the offloaded tasks by forming a mobile device cloud to eliminate the limitations of offloading data to the remote or edge servers. Even with different implementation characteristics, the role of these diverse mobile cloud-assisted platforms in mobile device energy management cannot be underestimated. In reality, numerous heterogeneous clouds could occur in the environments that have different characteristics. In this way, migrating the same application to the different cloud environments for execution could lead to different calculations and communication time due to different cloud characteristics and network environments at the same time. Hence, optimal cloud path offloading selection schemes are necessary to choose an energy-efficient cloud-assisted mobile platform for tasks execution. However, the selection process is complicated due to the need for analyzing various data and considering multiple factors such as bandwidth, server speed, security and privacy, and cloud availability. Table 10 gives a summary of the performance comparisons of cloud-assisted execution mechanisms.

7 CONCLUSIONS AND FUTURE CHALLENGES

In this article, a survey was presented on techniques for energy optimization of mobile devices. We reviewed and summarized energy management methods from perspectives of devices' hardware and software, computation offloading strategies, energy-efficient wireless data transmission, and cloud execution mechanisms for offloaded application components, respectively. We conclude this article with a brief summary of challenges in this area from standpoints of mobile hardware and software, computation offloading strategies in heterogeneous cloud environments, and complicated tradeoffs between multiple parties (e.g., energy efficiency, transmission delay, quality of service, and service pricing) involved in optimizing device energy consumption.

Challenges on mobile hardware and software design. Though mobile hardware has undergone revolutionary development, from initial support for phone service to diverse services and mobile applications, their capabilities such as storage, processing, and energy capacity are far from meeting the requirements of these ever changing services and mobile applications. **(1) Gap between hardware design and application requirements:** Given Moore's law, the semiconductor manufacturers need to spend at least 1.5 years to increase the amount of on-chip transistors by two times. However, it takes the battery manufacturers at least 5 years to achieve similar growth in power density, which is a far cry from the growth rate of the number of on-chip transistors. The increase in the volumetric/gravimetric energy density of the batteries lags behind the increased power demand of mobile devices, leading to a power crisis in mobile devices. **(2) Application and service design:** Usually, a well-designed mobile application or service could save significant power compared with an inefficient and poorly designed application or service. Hence, mobile application developers should leverage efficient energy-saving and performance-enhancing coding practices to design energy-aware mobile applications or services running on mobile devices. **(3) Summary:** The above discrepancy between energy density and performance improvements opens up a new horizon for researchers to explore the multifarious energy-saving opportunities for mobile devices by collaboratively optimizing energy of hardware and software to make up for the deficiency of limited resources on devices. For example, for computation/memory/communication-intensive mobile applications, designing application-oriented energy-aware mobile OSs and mobile hardware is urgently needed. Meanwhile, for application developers, fully exploring the execution characteristics and energy usage patterns of applications running on mobile OSs is helpful to effectively utilize the hardware resources on mobile devices.

Table 10. A Comparison Summary of Cloud Offloading Execution Mechanisms

Method	Goal	E_loc	D_e	T_e	C_e	Off_parti
CTE_MDP	RC	cloud clone	✓	★	×	offline
DME	RC	cloud server	✓	★	×	×
Clone2Clone	RC	cloud clone	✓	✓	×	×
TSGBO	RC	cloud server	✓	×	× (profit)	offline
EMOP	RC	cloud server	✓	✓	×	adaptive
FHMCO	RC	cloud server	✓	★	×	offline & online
ME-VoLTE	EC	edge server	✓	★	×	×
CCOA	EC	edge server	✓	×	×	×
ETEO	EC	edge server	✓	✓	×	offline
DOEHD	EC	edge server	✓	✓	×	×
MUMTO	EC	edge server	✓	✓	×	×
Serendipity	MDC	mobile device	✓	★	×	offline
M2MCO	MDC	mobile device	✓	★	×	×
MDCPM	MDC	mobile device	✓	×	×	×
FemtoClouds	HC	cloudlet&device	✓	★	×	
ENOA	HC	cloudlet&clone	✓	✓	×	×
DECM	HC	cloudlet&server	✓	✓	×	×
FAHV	HC	cloudlet&server	✓	★	×	×
CSOS	HC	MDC&EC&RC	✓	✓	×	runtime
MAPCloud	HC	local&public	✓	✓	×	×
GTCCO	HC	EC&RC	✓	✓	×	×

Method	Deci.	Gran.	Ene.	Perf.	Benchmark	Reference
CTE_MDP	dyn	task	+5x/+20%	÷	local_o/remote_o	[Zhang et al. 2015b]
DME	dyn	functionality	+60%	+79%	original version	[Kwon and Tilevic 2012]
Clone2Clone	×	communication	+30%	÷	SPORC	[Kosta et al. 2013]
TSGBO	dyn	service request	+21.8%	31.9%	local execution	[Wang et al. 2013]
EMOP	dyn	data/process	+30.8%	÷	single site	[Terefe et al. 2016]
FHMCO	dyn	task	+38%	+44%	MMRO	[Goudarzi et al. 2017]
ME-VoLTE	dyn	encoding effort	+13%	÷	local execution	[Beck et al. 2015]
CCOA	dyn	task	+45%	+45%	non cooperation	[Fan et al. 2017]
ETEO	dyn	task	+ 63%	+ 63%	local execution	[Zhang et al. 2017b]
DOEHD	dyn	task	+14%	÷	CPU_only	[Zhang et al. 2018]
MUMTO	dyn	task	+31%	+31%	random scheme	[Chen et al. 2018]
Serendipity	dyn	task	+74%	+6.6x	local execution	[Shi et al. 2012]
M2MCO	dyn	task	+50%	+4x	local execution	[Mtibaa et al. 2014]
MDCPM	×	×	+10%	+8x	learning method	[Tian et al. 2018]
FemtoClouds	dyn	task	+26%	+85%	PreOb	[Habak et al. 2015]
ENOA	dyn	app	+90%	+60%	local_o/clone_o	[Magurawalage et al. 2014]
DECM	dyn	task	+58%	÷	traditional method	[Gai et al. 2016]
FAHV	dyn	task	+236%	÷	cloud_only	[Ravi and Peddoju 2015]
CSOS	dyn	code	+55.6%	÷	local execution	[Zhou et al. 2015]

(Continued)

Table 10. Continued

Method	Deci.	Gran.	Ene.	Perf.	Benchmark	Reference
MAPCloud	dyn	task	+32%	+40%	public_o	[Rahimi et al. 2012]
GTCCO	dyn	task	+ 20%	÷	distributed_offload	[Guo and Liu 2018]

E_loc: Execution location, D_e: Device energy, T_e: Transmission energy, C_e: Cloud server energy, Off_parti: Offloading partition, Deci.: Decision, Gran.: Granularity, Ene.: Energy saving, Perf.: Performance improvement. ✓ indicates that the factor is considered in the literature. ⋆ implies that the literature mentions the factor but does not conduct an in-depth study of it. × indicates that the factor is not considered in the literature. ÷ implies that the literature does not give a clear explanation of the factor.

Challenges on offloading between devices and clouds. Computation offloading is a complex process since it involves collaborative operations among mobile devices, communication channels, and cloud servers. MCC and MEC face many technical challenges, such as user mobility, network bandwidth, cloud availability and heterogeneity, data security and privacy, making offloading difficult to perform and implement in real-world environments. **(1) Security, privacy, and trust:** A survey by Fujitsu Research found that 88% of potential cloud users are concerned about their data security and privacy in the clouds. For effective deployment of MCC/MEC on a global scale, cloud providers need to establish trustworthy mechanisms for its users. If cloud providers are not worthy of their users' trust, these users may refuse to use cloud services, then offloading will become useless. Service-level agreement (SLA) builds a trust mechanism between users and service providers by guaranteeing quality of service and data security in clouds. During offloading, the offloaded data in communication channels may face new threats, such as information leakage and malicious attacks. To this end, devices may leverage encryption technologies to encrypt offloaded data before data transfer for ensuring data security and privacy. However, encryption operations will bring additional energy consumption to devices. **(2) Dynamic wireless network conditions:** Dynamic wireless network conditions also pose energy optimization challenges when performing offloading from mobile devices. For example, poor wireless connectivity usually consumes greater device energy compared with good wireless connectivity. Thus, developing effective forward-looking wireless signal prediction algorithms is especially necessary. **(3) Heterogeneous cloud environment:** Heterogeneous cloud environment (e.g., remote clouds, edge/fog clouds, and mobile device clouds) are usually in the different contexts with different computing and storage capabilities (e.g., network bandwidth, cloud service price, server speed, and data security), also bringing difficulty for choosing a green cloud offloading path for mobile devices. **(4) User mobility and diverse application and service:** Due to the mobility of mobile devices, offloading failures may occur, resulting in unpredictable performance of programs. Further, the diversity of services and mobile applications also makes it intractable to design generic service or application partition mechanisms. **(5) Summary:** To achieve energy-efficient offloading, it is important to develop application-aware, wireless signal-aware, and server-aware dynamic offloading strategies to minimize mobile device energy consumption.

Challenges on tradeoffs among multiple parties for mobile energy optimization. The energy consumption of mobile devices can be caused by OS design, hardware design and management, application and service design, and wireless communication design. **(1) Application and hardware design tradeoff:** Currently, the design principle of the mobile applications and services just pursue the performance improvements and ignore the power limitation of mobile devices. Thus, when designing mobile OSs and mobile hardware components, researchers need to consider the requirements and the characteristics of the applications and the services. Meanwhile, when designing applications and services, developers should not only pursue the

performance improvements, but also consider the power crisis on mobile devices. Overall, designing and developing power-aware applications and services is urgently needed for devices. **(2) Application and transmission design tradeoff:** Different transmission technologies are designed with different performance metrics such as bandwidth and cellular coverage. Usually, standardized communication protocols often support unnecessary features for many applications, leading to unnecessary power waste for transferring the useless data required for HTTP transactions while application-specific communication protocols only support the specific dataset. Hence, balancing the design principle of general-purpose and dedicated-purpose communication protocols is essential for green data communication. **(3) Tradeoffs among various factors during offloading:** Offloading frees mobile devices from a power crisis; however, it is not free and could incur other costs and energy required for data communication. During offloading, factors such as network bandwidth, cloud service price, energy efficiency, transmission delay, user cost, server speed, and cloud availability have different effects on offloading decisions. For example, when the wireless network connectivity is stable, we can migrate vast data to the clouds for execution. Otherwise, only a small quantity of data can be migrated to the clouds for transmission energy savings. Cloud service price also has a direct influence on offloading decisions. For example, if the service price is set too high, users will refuse to use cloud services and, thus, cannot use offloading services. Hence, for service providers, designing an economical service delivery solution is critical. Moreover, if users expect to reduce communication latency and application execution time, they may choose wireless interfaces with high bandwidth and purchase servers with high configurations, but these operations will bring more energy consumption and capital expenditure. **(4) Summary:** Most existing works focus more on the balance between performance improvement and energy saving while ignoring other factors such as cloud service price, data security and privacy, and offloading costs when optimizing device energy consumption. How to balance the above multiple factors during computation offloading is a challenging and urgent problem to solve for the researchers who are interested in this area.

REFERENCES

2018. statista. https://www.statista.com/statistics/274774/forecast-of-mobile-phone-users-worldwide/.

2019. Communication protocol. https://en.wikipedia.org/wiki/Communication_protocol.

2019. Computation offloading. https://en.wikipedia.org/wiki/Computation_offloading.

2019. Edge computing. https://en.wikipedia.org/wiki/Edge_computing.

2019. Guglielmo Marconi. https://en.wikipedia.org/wiki/Guglielmo_Marconi.

2019. Mobile computing. https://en.wikipedia.org/wiki/Mobile_computing.

2019. Mobile device. https://en.wikipedia.org/wiki/Mobile_device.

2019. Mobile operating system. https://en.wikipedia.org/wiki/Mobile_operating_system.

2019. Wireless. https://en.wikipedia.org/wiki/Wireless.

T. Clausen, A. Augustin, J. Yi, and W. Townsley. 2016. A study of LoRa: Long range & low power networks for the Internet of Things. *Sensors* 16, 9 (2016), 1466.

M. Shafique, T. Mitra, A. Prakash, H. Amrouch, and J. Henkel. 2016. Improving mobile gaming performance through cooperative CPU-GPU thermal management. In *DAC* (2016), 1–6.

K. Akherfi, M. Gerndt, and H. Harroud. 2018. Mobile cloud computing for computation offloading: Issues and challenges. *Applied Computing and Informatics* 14, 1 (2018), 1–16.

M. Altamimi, R. Palit, K. Naik, and A. Nayak. 2012. Energy-as-a-Service (EaaS): On the efficacy of multimedia cloud computing to save smartphone energy. In *CLOUD* (2012), 764–771.

P. Balakrishnan and C. Tham. 2013. Energy-efficient mapping and scheduling of task interaction graphs for code offloading in mobile cloud computing. In *UCC* (2013), 34–41.

S. Barbarossa, S. Sardellitti, and P. Lorenzo. 2013. Computation offloading for mobile cloud computing based on wide cross-layer optimization. *Future Network and Mobile Summit* (2013), 1–10.

M. Barbera, S. Kosta, A. Mei, and J. Stefa. 2013. To offload or not to offload? The bandwidth and energy costs of mobile cloud computing. In *INFOCOM* (2013), 1285–1293.

M. Beck, S. Feld, and T. Schimper. 2015. ME-VoLTE: Network functions for energy-efficient video transcoding at the mobile edge. In *ICIN* (2015), 38–44.

D. Brooks, V. Tiwari, and M. Martonosi. 2000. Wattch: A framework for architectural-level power analysis and optimizations. *ACM SIGARCH Computer Architecture News* 28, 2 (2000), 83–94.

D. Bui, K. Lee, S. Oh, I. Shin, H. Shin, H. Woo, and D. Ban. 2013. GreenBag: Energy-efficient bandwidth aggregation for real-time streaming in heterogeneous mobile wireless networks. In *RTSS* (2013), 57–67.

A. Carroll and G. Heiser. 2010. An analysis of power consumption in a smartphone. In *USENIX ATC*, Vol. 14. 21.

W. Chen, S. Cheng, P. Hsiu, and T. Kuo. 2015. A user-centric CPU-GPU governing framework for 3D games on mobile devices. In *ICCAD* (2015), 224–231.

W. Chen, D. Wang, and K. Li. 2018. Multi-user multi-task computation offloading in green mobile edge cloud computing. *IEEE Transactions on Services Computing* 12, 5 (2018), 726–738.

X. Chen, L. Jiao, W. Li, and X. Fu. 2016. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Transactions on Networking* 24, 5 (2016), 2795–2808.

D. Chu, A. Kansal, J. Liu, and F. Zhao. 2011. Mobile Apps: It's time to move up to CondOS. In *HotOS* (2011), 16–16.

P. Chuang, Y. Chen, and P. Huang. 2017. An adaptive on-line CPU-GPU governor for games on mobile devices. In *ASP-DAC* (2017), 653–658.

B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. 2011. Clonecloud: Elastic execution between mobile device and cloud. In *EuroSys* (2011), 301–314.

B. Chun and P. Maniatis. 2010. Dynamically partitioning applications between weak devices and clouds. In *MCS* (2010), 1–5.

P. Cong, L. Li, J. Zhou, K. Cao, T. Wei, M. Chen, and S. Hu. 2018. Developing user perceived value based pricing models for cloud markets. *IEEE Transactions on Parallel and Distributed Systems* 29, 12 (2018), 2742–2756.

E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. 2010. MAUI: Making smartphones last longer with code offload. In *MobiSys* (2010), 49–62.

Y. Cui, X. Ma, H. Wang, I. Stojmenovic, and J. Liu. 2013. A survey of energy efficient wireless transmission and modeling in mobile cloud computing. *Mobile Networks and Applications* 18, 1 (2013), 148–155.

Y. Cun, J. Denker, and S. Sol1a. 1990. Optimal brain damage. In *NIPS* (1990), 598–605.

S. Deng and H. Balakrishnan. 2012. Traffic-aware techniques to reduce 3G/LTE wireless energy consumption. In *CoNext* (2012), 181–192.

A. Ding, B. Han, Y. Xiao, P. Hui, A. Srinivasan, M. Kojo, and S. Tarkoma. 2013a. Enabling energy-aware collaborative mobile data offloading for smartphones. In *SECON* (2013), 487–495.

N. Ding, D. Wagner, X. Chen, A. Pathak, Y. Hu, and A. Rice. 2013b. Characterizing and modeling the impact of wireless signal strength on smartphone battery drain. In *SIGMETRICS* (2013), 29–40.

F. Dogar, P. Steenkiste, and K. Papagiannaki. 2010. Catnap: Exploiting high bandwidth wireless interfaces to save energy for mobile devices. In *MobiSys* (2010), 107–122.

B. Donohoo, C. Ohlsen, S. Pasricha, and C. Anderson. 2012. Exploiting spatiotemporal and device contexts for energy-efficient mobile embedded systems. In *DAC* (2012), 1278–1283.

W. Fan, Y. Liu, B. Tang, F. Wu, and Z. Wang. 2017. Computation offloading based on cooperations of mobile edge computing-enabled base stations. *IEEE Access* 6 (2017), 22622–22633.

A. Feldman, W. Zeller, M. Freedman, and E. Felten. 2010. SPORC: Group collaboration using untrusted cloud resources. In *OSDI Symposia* 10 (2010), 337–350.

K. Gai, M. Qiu, H. Zhao, L. Tao, and Z. Zong. 2016. Dynamic energy-aware cloudlet-based mobile cloud computing model for green computing. *Journal of Network and Computer Applications* 59 (2016), 46–54.

Y. Ge, Y. Zhang, Q. Qiu, and Y. Lu. 2012. A game theoretic resource allocation for overall energy minimization in mobile cloud computing system. In *ISLPED* (2012), 279–284.

M. Goudarzi, M. Zamani, and A.Haghighat. 2017. A fast hybrid multi-site computation offloading for mobile cloud computing. *Journal of Network and Computer Applications* 80 (2017), 219–231.

H. Guo and J. Liu. 2018. Collaborative computation offloading for multiaccess edge computing over fiber-wireless networks. *IEEE Transactions on Vehicular Technology* 67, 5 (2018), 4514–4526.

K. Habak, M. Ammar, K. Harras, and E. Zegura. 2015. FemtoClouds: Leveraging mobile devices to provide cloud service at the edge. In *CLOUD* (2015), 9–16.

S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. Horowitz, and W. Dally. 2016a. EIE: Efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 243–254.

S. Han, H. Mao, and W. Dally. 2016b. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *ICLR* (2016).

S. Han, J. Pool, J. Tran, and W. Dally. 2015. Learning both weights and connections for efficient neural networks. In *NIPS* (2015), 1135–1143.

S. Hao, D. Li, W. Halfond, and R. Govindan. 2013. Estimating mobile application energy consumption using program analysis. In *ICSE* (2013), 92–101.

Y. He, X. Zhang, and J. Sun. 2017. Channel pruning for accelerating very deep neural networks. In *ICCV* (2017), 1389–1397.

M. Hoque, M. Siekkinen, J. Nurminen, S. Tarkoma, and M. Aalto. 2014. Saving energy in mobile devices for on-demand multimedia streaming-a cross-layer approach. *ACM Transactions on Multimedia Computing Communications, and Applications* 10, 3 (2014), 1–23.

C. Hsieh, J. Park, N. Dutt, and S. Lim. 2015. Memory-aware cooperative CPU-GPU DVFS governor for mobile games. In *ESTIMedia* (2015), 1–8.

Y. Jararweh, L. Tawalbeh, F. Ababneh, A. Khreishah, and F. Dosari. 2014. Scalable cloudlet-based mobile computing model. In *MobiSPC* 34 (2014), 434–441.

H. Jiang, H. Yang, S. Qin, Z. Su, J. Zhang, and J. Yan. 2017. Detecting energy bugs in android apps using static analysis. In *ICFEM* (2017), 192–208.

D. Kadjo, R. Ayoub, M. Kishinevsky, and P. Gratz. 2015. A control-theoretic approach for energy efficient CPU-GPU subsystem in mobile platforms. In *DAC* (2015), 1–6.

G. Kalic, I. Bojic, and M. Kusek. 2012. Energy consumption in Android phones when using wireless communication technologies. In *MIPRO* (2012), 754–759.

R. Kemp, N. Palmer, T. Kielmann, and H. Bal. 2010. Cuckoo: A computation offloading framework for smartphones. In *MobiCASE* (2010), 59–79.

D. Kim, N. Jung, and H. Cha. 2014. Content-centric display energy management for mobile devices. In *DAC* (2014), 1–6.

Y. Kim, F. Parterna, S. Tilak, and T. Rosing. 2015. Smartphone analysis and optimization based on user activity recognition. In *ICCAD* (2015), 605–612.

M. Kjærgaard, S. Bhattacharya, H. Blunck, and P. Nurmi. 2011. Energy-efficient trajectory tracking for mobile devices. In *MobiSys* (2011), 307–320.

S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang. 2012. ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *INFOCOM* (2012), 945–953.

S. Kosta, V. Perta, J. Stefa, P. Hui, and A. Mei. 2013. Clone2Clone (C2C): Peer-to-peer networking of smartphones on the cloud. In *HotCloud* (2013).

D. Kovachev, T. Yu, and R. Klamma. 2012. Adaptive computation offloading from mobile devices into the cloud. In *ISPA* (2012), 784–791.

K. Kumar, J. Liu, Y. Lu, and B. Bhargava. 2013. A survey of computation offloading for mobile systems. *Mobile Networks and Applications* 18, 1 (2013), 129–140.

K. Kumar and Y. Lu. 2010. Cloud computing for mobile users: Can offloading computation save energy? *Computer* 43, 4 (2010), 51–56.

Y. Kwon and E. Tilevic. 2012. Power-efficient and fault-tolerant distributed mobile execution. In *ICDCS* (2012), 586–595.

Y. Kwon and E. Tilevich. 2013. Reducing the energy consumption of mobile applications behind the scenes. In *ICSM* (2013), 170–179.

R. Lange, T. Farrell, F. Dürr, and K. Rothermel. 2009. Remote real-time trajectory simplification. In *PerCom* (2009), 1–10.

W. Lee, Y. Wang, D. Shin, N. Chang, and M. Pedram. 2014. Optimizing the power delivery network in a smartphone platform. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33, 1 (2014), 36–49.

D. Li and W. Halfond. 2014. An investigation into energy-saving programming practices for Android smartphone app development. In *GREENS* (2014), 46–53.

D. Li, S. Hao, W. Halfond, and R. Govindan. 2013b. Calculating source line level energy information for Android applications. In *ISSTA* (2013), 78–89.

J. Li, K. Bu, X. Liu, and B. Xiao. 2013a. ENDA: Embracing network inconsistency for dynamic application offloading in mobile cloud computing. In *SIGCOMM MCC* (2013), 39–44.

X. Li, G. Yan, Y. Han, and X. Li. 2013c. Smartcap: User experience-oriented power adaptation for smartphone's application processor. In *DATE* (2013), 57–60.

Z. Li, C. Wang, and R. Xu. 2001. Computation offloading to save energy on handheld devices: A partition scheme. In *CASES* (2001), 238–246.

Y. Liang, X. Zhou, Z. Yu, and B. Guo. 2014. Energy-efficient motion related activity recognition on mobile devices for pervasive healthcare. *Mobile Networks and Applications* 19, 3 (2014), 303–317.

K. Lin, A. Kansal, and D. Lymberopoulos. 2010. Energy-accuracy trade-off for continuous mobile device location. In *MobiSys* (2010), 285–298.

X. Lin, Y. Wang, and M. Pedram. 2013. An optimal control policy in a mobile cloud computing system based on stochastic data. In *CloudNet* (2013), 117–122.

X. Lin, Y. Wang, Q. Xie, and M. Pedram. 2014. Task scheduling with dynamic voltage and frequency scaling for energy minimization in the mobile cloud computing environment. *IEEE Transactions on Services Computing* 8, 2 (2014), 175–186.

F. Liu, P. Shu, H. Jin, L. Ding, J. Yu, D. Niu, and B. Li. 2013a. Gearing resource-poor mobile devices with powerful clouds: Architectures, challenges, and applications. *IEEE Wireless Communications* 20, 3 (2013), 14–22.

J. Liu, K. Kumar, and Y. Lu. 2010. Tradeoff between energy savings and privacy protection in computation offloading. In *ISLPED* (2010), 213–218.

T. Liu, F. Chen, Y. Ma, and Y. Xie. 2016. An energy-efficient task scheduling for mobile devices based on cloud assistant. *Future Generation Computer Systems* 61 (2016), 1–12.

Y. Liu, C. Xu, and S. Cheung. 2013b. Where has my battery gone? Finding sensor related energy black holes in smartphone applications. In *PerCom* (2013), 2–10.

T. Luan, L. Gao, Z. Li, Y. Xiang, G. We, and L. Sun. 2015. Fog computing: Focusing on mobile users at the edge. *[Online]*. Retrieved 2016 from https://arxiv.org/abs/1502.01815.

A. Maghazeh, U. Bordoloi, M. Villani, P. Eles, and Z. Peng. 2015. Perception-aware power management for mobile games via dynamic resolution scaling. In *ICCAD* (2015), 613–620.

C. Magurawalage, K. Yang, L. Hu, and J. Zhang. 2014. Energy-efficient and network-aware offloading algorithm for mobile cloud computing. *Computer Networks* 74 (2014), 22–33.

D. McIntire, K. Ho, B. Yip, A. Singh, W. Wu, and W. Kaiser. 2006. The low power energy aware processing (LEAP) embedded networked sensor system. In *IPSN* (2006), 449–457.

E. Miluzzo, R. Caceres, and Y. Chen. 2012. Vision: mClouds-computing on clouds of mobile devices. In *MCS* (2012), 9–14.

A. Mtibaa, A. Fahim, K. Harras, and M. Ammar. 2013a. Towards resource sharing in mobile device clouds: Power balancing across mobile devices. *ACM SIGCOMM Computer Communication Review* 43, 4 (2013), 51–56.

A. Mtibaa, K. Harras, and A. Fahim. 2013b. Towards computational offloading in mobile device clouds. In *CloudCom* (2013), 331–338.

A. Mtibaa, M. Snober, A. Carelli, R. Beraldi, and H. Alnuweiri. 2014. Collaborative mobile-to-mobile computation offloading. In *CollaborateCom* (2014), 460–465.

V. Namboodiri and T. Ghose. 2012. To cloud or not to cloud: A mobile device perspective on energy consumption of applications. In *WoWMoM* (2012), 1–9.

Q. Nguyen, J. Blobel, and F. Dressler. 2016. Energy consumption measurements as a basis for computational offloading for android smartphones. In *CSE* (2016), 24–31.

S. Nirjon, A. Nicoara, C. Hsu, J. Singh, and J. Stankovic. 2012. MultiNets: Policy oriented real-time switching of wireless interfaces on mobile devices. In *RTAS* (2012), 251–260.

J. Paek, J. Kim, and R. Govindan. 2010. Energy-efficient rate-adaptive GPS-based positioning for smartphones. In *MobiSys* (2010), 299–314.

V. Pallipadi and A. Starikovskiy. 2006. The ondemand governor. In *Proceedings of Linux Symposium* 2, 00216 (2006), 215–230.

F. Paterna and T. Rosing. 2015. Modeling and mitigation of extra-SoC thermal coupling effects and heat transfer variations in mobile devices. In *ICCAD* (2015), 831–838.

A. Pathak, Y. Hu, and M. Zhang. 2011. Bootstrapping energy debugging on smartphones: A first look at energy bugs in mobile devices. In *HotNets* (2011), 1–6.

A. Pathak, Y. Hu, and M. Zhang. 2012. Where is the energy spent inside my app? Fine grained energy accounting on smartphones with eprof. In *EuroSys* (2012), 29–42.

A. Pathania, A. Irimiea, A. Prakash, and T. Mitra. 2015. Power-performance modelling of mobile gaming workloads on heterogeneous MPSoCs. In *DAC* (2015), 1–6.

A. Pathania, Q. Jiao, A. Prakash, and T. Mitra. 2014. Integrated CPU-GPU power management for 3D mobile games. In *DAC* (2014), 1–6.

C. Pluntke, L. Eggert, and N. Kiukkonen. 2011. Saving mobile device energy with multipath TCP. In *MobiArch* (2011), 1–6.

B. Priyantha, D. Lymberopoulos, and J. Liu. 2011. Little rock: Enabling energy efficient continuous sensing on mobile phones. *IEEE Pervasive Computing* 10, 2 (2011), 12–15.

H. Qian and D. Andresen. 2014. Jade: An efficient energy-aware computation offloading system with heterogeneous network interface bonding for ad-hoc networked mobile devices. In *SNPD* (2014), 1–8.

N. Mangalvedhe R. Ratasuk, B. Vejlgaard and A. Ghosh. 2016. NB-IoT system for M2M communication. In *WCNC* (2016), 1–5.

Y. Wei R. Sinha and S. Hwang. 2017. A survey on LPWA technology: LoRa and NB-IoT. *ICT Express* 3, 1 (2017), 14–21.

M. Rahimi, N. Venkatasubramanian, S. Mehrotra, and A. Vasilakos. 2012. MAPCloud: Mobile applications on an elastic and scalable 2-tier cloud architecture. In *UCC* (2012), 83–90.

A. Ravi and S. Peddoju. 2015. Handoff strategy for improving energy efficiency and cloud service availability for mobile devices. *Wireless Personal Communications* 81, 1 (2015), 101–132.

I. Rodero and et al.2010. Towards energy-efficient reactive thermal management in instrumented datacenters. In *GRID* (2010), 321–328.

N. Rodriguez and J. Crowcroft. 2011. ErdOS: Achieving energy savings in mobile OS. In *MobiArch* (2011), 37–42.

N. Rodriguez and J. Crowcroft. 2013. Energy management techniques in modern mobile handsets. *IEEE Communications Surveys & Tutorials* 15, 1 (2013), 179–198.

A. Roy, S. Rumble, R. Stutsman, P. Levis, D. Mazieres, and N. Zeldovich. 2011. Energy management in mobile devices with the cinder operating system. In *EuroSys* (2011), 139–152.

M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. 2009. The case for VM-based cloudlets in mobile momputing. *IEEE Pervasive Computing* 8 (2009), 14–23.

C. Shi, K. Habak, P. Pandurangan, M. Ammar, M. Naik, and Ellen Zegura. 2014. COSMOS: Computation offloading as a service for mobile devices. In *MobiHoc* (2014), 287–296.

C. Shi, V. Lakafosis, M. Ammar, and E. Zegura. 2012. Serendipity: Enabling remote computing among intermittently connected mobile devices. In *MobiHoc* (2012), 145–154.

W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. 2016. Edge computing: Vision and challenges. *IEEE Internet of Things Journal* 3, 5 (2016), 637–646.

M. Shiraz and A. Gani. 2014. A lightweight active service migration framework for computational offloading in mobile cloud computing. *The Journal of Supercomputing* 68, 2 (2014), 978–995.

P. Shu, F. Liu, H. Jin, M. Chen, F. Wen, Y. Qu, and B. Li. 2013. eTime: Energy-efficient transmission between cloud and mobile devices. In *INFOCOM* (2013), 195–199.

G. Singla, G. Kaur, A. Unver, and U. Ogras. 2015. Predictive dynamic thermal and power management for heterogeneous mobile platforms. In *DATE* (2015), 960–965.

M. Terefe, H. Lee, N. Heo, G. Fox, and S. Oh. 2016. Energy-efficient multisite offloading policy using Markov decision process for mobile cloud computing. *Pervasive and Mobile Computing* 27 (2016), 75–89.

Z. Tian, Z. Wang, H. Li, P. Yang, R. Maeda, and J. Xu. 2018. Multi-device collaborative management through knowledge sharing. In *ASP-DAC* (2018), 22–27.

L. Tong and W. Gao. 2016. Application-aware traffic scheduling for workload offloading in mobile clouds. In *INFOCOM* (2016), 1–9.

P. Kulkarni U. Raza and M. Sooriyabandara. 2017. Low power wide area networks: An overview. *IEEE Communications Surveys & Tutorials* 19, 6 (2017), 855–873.

T. Wang, J. Zhou, G. Zhang, T. Wei, and S. Hu. 2020. Customer perceived value- and risk-aware multiserver configuration for profit maximization. *IEEE Transactions on Parallel and Distributed Systems* 31, 5 (2020), 1074–1088.

Y. Wang, X. Lin, and M. Pedram. 2013. A nested two stage game-based optimization framework in mobile cloud computing system. In *SOSE* (2013), 494–502.

G. Welch. 1995. A survey of power management techniques in mobile computing operating systems. *ACM SIGOPS Operating Systems Review* 29, 4 (1995), 47–56.

H. Wu and D. Huang. 2014. Modeling multi-factor multi-site risk-based offloading for mobile cloud computing. In *CNSM* (2014), 230–235.

H. Wu and K. Wolter. 2015. Analysis of the energy-performance tradeoff for delayed mobile offloading. In *Valuetools* (2015), 250–258.

Z. Wu, L. Gui, J. Chen, H. Zhou, and F. Hou. 2016. Mobile cloudlet assisted computation offloading in heterogeneous mobile cloud. In *WCSP* (2016), 1–6.

F. Xia, F. Ding, J. Li, X. Kong, L. Yang, and J. Ma. 2014. Phone2Cloud: Exploiting computation offloading for energy saving on smartphones in mobile cloud computing. *Information Systems Frontiers* 16, 1 (2014), 95–111.

C. Xian, Y. Lu, and Z. Li. 2007. Adaptive computation Offloading for energy conservation on battery-powered systems. In *ICPADS* (2007), 1–8.

Y. Xiao. 2011. Modeling and managing energy consumption of mobile devices. *Aalto University Publication* (2011).

K. Yaghmour. 2012. Inside Android's UI. In *ELCE* (2012).

Z. Yan, V. Subbaraju, D. Chakraborty, A. Misra, and K. Aberer. 2012. Energy-efficient continuous activity recognition on mobile phones: An activity-adaptive approach. In *ISWC* (2012), 17–24.

L. Yang, R. Dick, G. Memik, and P. Dinda. 2013a. HAPPE: Human and application-driven frequency scaling for processor power efficiency. *IEEE Transactions on Mobile Computing* 12, 8 (2013), 1546–155.

S. Yang, Y. Kwon, Y. Cho, H. Yi, D. Kwon, J. Youn, and Y. Paek. 2013b. Fast dynamic execution offloading for efficient mobile cloud computing. In *PerCom* (2013), 20–28.

D. Yao, C. Yu, H. Jin, and J. Zhou. 2013. Energy efficient task scheduling in mobile cloud computing. In *NPC* (2013), 344–355.

J. Yoo and K. Park. 2011. A cooperative clustering protocol for energy saving of mobile devices with WLAN and Bluetooth interfaces. *IEEE Transactions on Mobile Computing* 10, 5 (2011), 491–504.

G. Zhang, W. Zhang, Y. Cao, D. Li, and L. Wang. 2018. Energy-delay tradeoff for dynamic offloading in mobile-edge computing system with energy harvesting devices. *IEEE Transactions on Industrial Informatics* 14, 10 (2018), 4642–4655.

J. Zhang, X. Hu, Z. Ning, E. Ngai, L. Zhou, J. Wei, J. Cheng, and B. Hu. 2017b. Energy-latency trade-off for energy-aware offloading in mobile edge computing networks. *IEEE Internet of Things Journal* 5, 4 (2017), 2633–2645.

L. Zhang, D. Fu, J. Liu, E. Ngai, and W. Zhu. 2017a. On energy-efficient offloading in mobile cloud for real-time video applications. *IEEE Transactions on Circuits and Systems for Video Technology* 27, 1 (2017), 170–181.

W. Zhang, Y. Wen, K. Guan, D. Kilper, H. Luo, and D. Wu. 2013. Energy-optimal mobile cloud computing under stochastic wireless channel. *IEEE Transactions on Wireless Communications* 12, 9 (2013), 4569–4581.

W. Zhang, Y. Wen, and D. Wu. 2015b. Collaborative task execution in mobile cloud computing under a stochastic wireless channel. *IEEE Transactions on Wireless Communications* 14, 1 (2015), 81–93.

Y. Zhang, D. Niyato, and P. Wang. 2015a. Offloading in mobile cloudlet systems with intermittent connectivity. *IEEE Transactions on Mobile Computing* 14, 12 (2015), 2516–2529.

B. Zhao, Q. Zheng, G. Cao, and S. Addepalli. 2013. Energy-aware web browsing in 3G based smartphones. In *ICDCS* (2013), 165–175.

B. Zhou, A. Dastjerdi, R. Calheiros, S. Srirama, and R. Buyya. 2015. A context sensitive offloading scheme for mobile cloud computing service. In *CLOUD* (2015), 869–876.

Received October 2018; revised November 2019; accepted January 2020